UKHEC

Data Management Tools for High Performance Computing Applications

J.V. Ashby and C. Greenough

Computational Science and Engineering Department
CLRC Rutherford Appleton Laboratory,
Chilton, Didcot OX11 0QX, UK

R.J. Allan

Computational Science and Engineering Department
CLRC Daresbury Laboratory, Warrington, WA4 4AD, UK

Email: `j.v.ashby@rl.ac.uk`, `c.greenough@rl.ac.uk` or `r.j.allan@dl.ac.uk`

August 2, 2001

Abstract

The management of data produced by experimental facilities, by remote sensing and by large scale simulations has, in recent years, become an increasingly important issue. There are many groups investigating different approaches, each developing new representation models and new software.

This report brings together information and some examples of the more commonly used tools available for the *managing* task of data management. There are perhaps hundreds of software tools available of which only a fraction is presented here. We have concentrated on the better known ones and those that are accepted by the scientific community as *de facto* standards.

Keywords: data management, software tools, High Performance Computing.

This is a Technical Report of the UKHEC Collaboration.

Report available from `http://www.ukhec.ac.uk/publications`

Contents

1 Introduction

The management of data produced by experimental facilities, by remote sensing and by large scale simulations has, in recent years, become an important issue. There are many groups investigating different approaches, each developing new representation models and new software.

The area of High End Computing has a variety of particular requirements. Many High Performance Computing (HPC) applications process data collected from experiments or remote sensing instruments. The volumes can range from a few Megabytes to Petabytes. HPC applications in their own right are generators of large quantities of data that are often required for check-pointing and re-start a computation or for subsequent analysis, including visualisation and virtual reality (VR).

The *organisation* of these vast collections of data is the science of data management. A common way of defining data management is to divide it into the tasks of capturing, storing, managing, analysing and visualising. Each of these tasks requires standards, techniques and software. A survey of the international background to data management and requirements of difference scientific disciplines is provided in [1].

The tasks of *data management* comprise:

Capture - moving the data from the instrument or simulation to the storage mechanism. This often involves some form of data selection or compression. It also involves the creation of metadata.

Storage - where and how the data is stored physically. The use of automatated data vaults and tape stores is commonplace.

Management - indexing and cataloguing the data and providing methods to organise and move it from site to site or between programs. This involves metadata and self-defining data formats.

Analysis - processing or fusion and mining the data to extract scientific insight.

Visualisation and VR - presenting the data in a variety of forms to aid analysis and the dissemination of results.

Another related area of research and development is that of parallel or distributed I/O systems and databases, see [8]. Although these could be seen as another element of data management we have chosen to try and distinguish the two.

This report only really looks at the *management* of data and then only from the point of view of an applications programmer. In many of the HPC applications the data has been captured, pre-analyised and stored in some format. The problems of scientists using this data is much more about access and transformation to more useful forms. For those who are generating data through simulation, the major problems are in how to store and manage the results for re-use and making it easy to catalogue, retrieve and available for future projects.

For these groups the systems available for storage and management are of interest. The physical and hardware solutions are not of so much interest but the functionality and efficiency of management tools is. However, both of these groups have an interest in tools for analysis and visualisation.

This report brings together information and some examples of the more common tools available for the *managing* task of data management for individual file-based data sets. There are perhaps hundreds of software tools available of which only a fraction is presented here. We have concentrated on the better known ones and those that are acceptance by the scientific community as *de facto* standards.

The remaining report is in two parts. For each tool we present a brief overview in Section 2 with contact details and, where available, a URL for more information or to download the tool. In addition, Section 3 supplements this with a user's investigation and comparison of some tools. As further assessments are done, tool descriptions will be moved from Section 2 to Section 3 in later versions of this report.

2 Data Management Formats and Tools

The information in this section has been obtained mainly from the Internet sources. Three sites have proven to be particularly useful:

- Ilana Stern's Scientific Data Format Information FAQ. The latest version of this list can be accessed at
`ftp://rtfm.mit.edu/pub/news.answers/sci-data-formats`.
- The Standards and Specifications List from the EC Diffuse project:
`http://www.diffuse.org`.
- NASA Data Assimilation Office:
`http://deo.gsfc.nasa.org`.

Where available the sections below contain a short description of the DM tool, an approriate URL and some contact details.

2.1 AFS - Andrews File System

Area covered:	file system management
Sponsors:	TRANSARC
Description:	AFS was originally developed at Carnegie Mellon University.

It is a network file system. Machines supporting AFS are called AFS clients. This concept is especially suited to solve the problem of multiple home directories. Users who have access to more than one computer tend to have multiple home directories – one on each machine they log on to. Under AFS, users always access the same home directory, no matter what AFS client they log on to.
Like the usual UNIX file system, the AFS file system is a tree-like system, however file names are not bound to a specific computer. Instead, file names within AFS are unique worldwide. They start with the string /afs and they contain the name of the AFS cell.
An AFS cell is the administrative domain and consists of a set of servers and clients.
AFS comprises of:

- file servers for AFS home directories and application programs;

- scalability: read-only data, i.e. application programs, can easily be replicated over several servers to distribute the load and guarantee service even if one server crashes;
- a local AFS cache on each participating host, providing a local copy of the files currently in use on this host;
- various other services, e.g. a Kerberos authentication server for better security.

From the user's point of view, an AFS system is identical on all participating hosts. The local cache exists for performance reasons only and is completely transparent to the users. They will see the same home directory on all AFS clients.

URL:	`http://www.transarc.com/Product/EFS/AFS/afsoverview.html`
Contacts:	TRANSARC 145 Cannon Street London EC4N 5BP

2.2 CDF – Common Data Format

Area covered:	Storing, manipulating and accessing multi-dimensional data sets.
Sponsors:	US National Space Science Data Center.
Description:	CDF (Common Data Format) is a library and toolkit for storing, manipulating, and accessing multi-dimensional data sets. The basic component of CDF is a software programming interface that is a device-independent view of the CDF data model. The CDF data model is based on the propagation of multi-dimensional arrays (grids) that classify data into variables that correspond to a single observable parameter. A CDF data set conforms to a basic grid structure where each variable has a position defined by indices. The CDF "Z structure" allows variables to be defined either independently of a grid or with respect to other variables. A data dictionary is used to uniquely identify variables. CDF data libraries support two physical encoding schemes: that of the underlying operating system; and the IEEE-754 format. The latter format provides a common data interchange format for CDF data. Developed by NASA as part of the US space program, CDF is widely used for interchange of scientific databases.
Contacts:	NASA Goddard Space Flight Center, Greenbelt, MD 20771, USA
URL:	`http://nssdc.gsfc.nasa.gov/cdf/cfd\_home.html`
FTP:	`ftp://nssdc.gsfc.nasa.gov/pub/cdf`

2.3 NetCDF - Network Common Data Format

Area covered:	Self-describing, network-transparent objects
Sponsors:	University Cooperation for Atmospheric Research (UCAR)
Description:	NetCDF is an interface for scientific data access which implements a machine-independent, self-describing, extendible file format. NetCDF is an abstraction that supports a view of data as a collection of self-describing, network-transparent objects that can be accessed through

a simple interface. Collections of named multi-dimensional variables can be randomly accessed, without knowing details of how the data are stored. Auxiliary information about the data, such as what units are used, can be stored with the data. Generic utilities and application programs can be written that access arbitrary NetCDF files and transform, combine, analyse, or display specified fields from the data.
A visual browser for netCDF format data files is available from `ftp://ftp.unidata.ucar.edu/pub/netcdf/contrib/ncview.tar.Z`.

Contacts:	UCAR Unidata Program Center P.O. Box 3000, Boulder, Colorado, USA 80307. Phone: +1 (303) 497-8644
URL:	Information on NetCDF can be obtained from `http://www.unidata.ucar.edu/packages/netcdf/`.
FTP:	Source code and documentation for the NetCDF data access library is available from `ftp://ftp.unidata.ucar.edu/pub/netcdf`.
Other information:	A recent paper by Jenter and Signell provides a good introduction to netCDF and is available as `ftp://crusty.er.usgs.gov/pub/netcdf.asce.ps`.

2.4 Deva – a STEP compliant database

Area covered:	Exchange of Engineering and Scientific Data
Sponsors:	CLRC Rutherford Appleton Laboratory
Description:	Deva is a database system which is based on the ISO 10303 standard STEP (Standard for the Exchange of Product model data) [9]. The data model can be defined through EXPRESS, the formal data modelling language defined as part of the STEP standard, and two APIs are available, a set of native Deva routines and a STEP Data Application Interface (SDAI) layer. The EXPRESS language describes constraints as well as data structure. Formal correctness rules will prevent conflicting interpretations. STEP CASE tools such as ST-Developer use these descriptions to create more robust, maintainable systems, see [9].
Contacts:	*J.V.Ashby@rl.ac.uk* or *C.Greenough@rl.ac.uk*
Reference:	[4]. For an example see Section 3.2.

2.5 DFS – Distributed File System

Area covered: Distributed file management.

Sponsors: MicroSoft.

Description: DFS allows system administrators to make it easier for users to access and manage files that are physically distributed across a network. With DFS, you can make files which are distributed across multiple servers appear to users as if they reside in one place on the network.
Users no longer need to know and specify the actual physical location of files in order to access them. Feature of Dfs are:

- Easy access to files – users need only go to one location on the network to access files, even though the files may be physically spread across multiple servers.

- Availability – domain-based Dfs ensures that users retain access to their files
- Server load balancing – a Dfs root can support multiple Dfs shared folders that are physically distributed across a network.

The Dfs client component runs on a number of different Windows platforms.

URL:	Information is available from `http://www.windows.com/windows2000/en/server/help/sag\_DFconceptOver.htm`
Contacts:	Microsoft Corporation

2.6 FITS – Flexible Image Transport System

Area covered:	Standard data interchange for the astronomy community.
Description:	FITS is the standard data interchange and archival format of the worldwide astronomy community. The NOST Standard and User's Guide, some software, and test files are available from `ftp://fits.gsfc.nasa.gov`. The site `http://fits.cv.nrao.edu` has other software and a different set of test files, and electronic copies of FITS proposals that are under development or in the review process. Archives of the USENET newsgroups `sci.data.formats`, `sci.astro.fits`, and others that are of interest to astronomers can also be found there.
Contacts:	NASA Goddard Space Flight Center, Greenbelt, MD 20771, USA
FTP:	`ftp://fits.cv.nrao.edu/fits`

2.7 GASS – Global Access to Secondary Storage

Area covered:	Distributed file system management on a computational Grid.
Sponsors:	Globus is a joint project of Argonne National Laboratory and the University of Southern California's Information Sciences Institute.
Description:	GASS provides a uniform name space (via URLs) and access mechanisms for files accessed via different protocols and stored in divers storage system types (HTTP, FTP, HPSS, DPSS, etc.); GASS is just one component of the Globus project which is developing basic middle-ware for the Grid that integrates geographically distributed computational and information resources, see `http://www.globus.org`.
URL:	`http://www.globus.org`
Contacts:	See `http://www.globus.org`
References:	Separate reports on computational Grid technology describing Globus are available [5, 6, 7].

2.8 GRIB – GRid In Binary

Area covered:	Storage of weather product information and exchange of weather product messages in gridded binary format.

Sponsors: World Meteorological Organization (WMO).

Description: GRIB is the WMO standard for gridded meteorological data.
Unfortunately it is still not very "standard", as some organizations use their own versions.
GRIB has a very efficient domain-specific compression algorithm.
Each GRIB record intended for either transmission or storage contains a single parameter with values located at an array of grid points, or represented as a set of spectral coefficients, for a single level (or layer), encoded as a continuous bit stream. Logical divisions of the record are designated as "sections", each of which provides control information and/or data.
A GRIB record consists of six sections, two of which are optional:

1. Indicator Section
2. Product Definition Section
3. Grid Description Section - optional
4. Bit Map Section - optional
5. Binary Data Section
6. ASCI termination characters "7777"

A formal description for WMO GRIB, and software to read general GRIB grids, can be found at `ftp://ncardata.ucar.edu/libraries/grib`. The format description can also be found at
`ftp://nic.fb4.noaa.gov/pub/nws/nmc/docs/gribguide/guide.txt`.
If you need GRIB to read ECMWF data, the above format description, along with the ECMWF-specific parameter table and a list of differences between the WMO and the ECMWF versions of GRIB, is in
`http://www-imk.fzk.de:8080/imk2/kasima/aktuelles/grid`.
Code for reading Grib files can be found in
`ftp://ncardata.ucar.edu/datasets/ds111.2/software`.

URL: `http://doe.gsfc.nasa.gov/data-stuff/formatPages/GRIB.html`

FTP: `ftp://ncardata.ucar.edu/libraries/grib`

Contacts: World Meteorological Organization, Geneva, Switzerland.

2.9 HDF – Hierarchical Data Format

Area covered: Interchange of scientific databases.

Sponsors: National Center for Supercomputing Applications (NCSA) .

Description: HDF is a self-defining file format for transfer of various types of data between different machines. The HDF library contains interfaces for storing and retrieving compressed or uncompressed raster images with palettes and an interface for storing and retrieving n-dimensional scientific datasets together with information about the data, such as labels, units, formats, and ranges for all dimensions.
HDF has been used in a number of application areas, including oceanographic modelling and computational chemistry and converters exist to import and export files in other formats. HDF is also acceptable to some graphics packages. It is available at some national data centres in the USA.

Version 3.3 of HDF allows JPEG images to be included in HDF data streams and supports the use of NetCDF. HDF allows meta-data about scientific data sets to be interchanged to provide details of:

- the coordinate system used when interpreting or displaying the data;
- ranges to be used for each dimension;
- labels for each dimension and the dataset as a whole;
- units for each dimension and the data;
- the valid maximum and minimum values for the data;
- calibration information for the data;
- fill or missing value information.

URL: `http://hdf.ncsa.uivc.edu`

FTP: `ftp://hdf.ncsa.uivc.edu/HDF5`

Contacts: NSCA, University of Illinois, Urbana-Champaign, USA.

2.10 HDS – Hierarchical Data System

Area covered: Scientific databases. It is presently used in astronomy, for storing (in particular) images, spectra and time series.

Description: HDS is a flexible system for storing and retrieving data and takes over from a computer's filing system at the level of an individual file. A conventional file effectively contains an 1-dimensional sequence of data elements, whereas an HDS file can contain a more complex structure. There are many parallels between the hierarchical way HDS stores data within files and the way that a filing system organises the files themselves. The advantage of HDS is that it allows many different kinds of data to be stored in a consistent and logical fashion. It is also very flexible, in that objects can be added or deleted whilst retaining the logical structure. HDS also provides portability of data, so that the same data objects may be accessed from different types of computer despite the fact that each may actually format its files and data in different ways.
HDS is a freely available database system. It is particularly suited to the storage of large multi-dimensional arrays (with their ancillary data) where efficiency of access is a requirement.

URL: `http://star-www.rl.ac.uk`

FTP: `ftp://starlink-ftp.rl.ac.uk/pub/doc/star-docs/sun92.tex`.

Contacts: StarLink Project,
CLRC Rutherford Appleton Lab,
Chilton, Dicot, Oxfordshire OX11 0QX

2.11 PDS – Planetary Data System

Area covered: Archiving space mission data on CD-ROM.

Sponsorings: NASA/Jet Propulsion Laboratory.

Description: Specification of formats used to store space mission data on CD-ROM. Contains specifications for:

- Cartographic Standards;
- Data Type Definitions;
- Data Products;
- Data Products Labels;
- Data Set/Data Set Collection Contents and Naming;
- Date/Time Format;
- Directory Types and Naming;
- Documentation Standard;
- File Specification and Naming;
- Media Formats for Data Submission and Archive;
- Object Description Language (ODL) Specification and Usage;
- PDS Objects;
- Pointer Usage;
- Record Formats;
- SFDU Usage;
- Usage of N/A, UNK and NULL;
- Units of Measurement;
- Volume Organization and Naming.

In recent years, the Planetary Data System has been used for archiving space mission data on CD-ROM media, using its own self-describing data format, variously know as PDS or ODL (Object Description Language). At least some of the current projects (e.g. Magellan, Galileo) are using the PDS format as a "pointer" to detached VICAR-format [10] imagery on the mission CD-ROM volumes.

URL:	`http://pds.jpl.nasa.gov/stdref/stdref.htm`.
Contacts:	Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, USA `pds_operator@jplpds.jpl.nasa.gov`.

2.12 RALBIC – A Neutral File Format for Finite Element Data

Area covered:	Interchange of finite element data, particularly in electromagnetics and semiconductor device analysis.
Sponsors:	CLRC Rutherford Appleton Laboratory
Description:	RALBIC is a well-defined neutral file format for the transfer of finite element (and finite volume) data. It is supported by a library of read and write routines written in FORTRAN77 and these provide access to commonly used data aggregates such as nodal coordinates, element topologies, nodal solutions, etc. RALBIC files can be either plain ASCII or binary.
Contacts:	`J.V.Ashby@rl.ac.uk` or `C.Greenough@rl.ac.uk`
Reference:	[3]. For an example see Section 3.1.

2.13 SAIF – Spatial Archive and Interchange Format

Area covered:	Geographic data exchange.
Sponsors:	Canadian General Standards Board, Committee on Geomatics.
Description:	SAIF is a Canadian standard for the exchange of geographic data. It uses an object oriented data model, and consists of definitions of the underlying building blocks, including tuples, sets, lists, enumerations and primitives.
URL:	`http://s2k-ftp.cs.berkeley.edu/pub/sequoia/schema/html/saif/saifHome.html`
FTP:	`http://s2k-ftp.cs.berkeley.edu/pub/sequoia/schema/STANDARDS/SAIF`
Contacts:	Surveys and Resource Mapping Branch, B.C. Minstry of Environment, Lands and Parks, Victoria, BC V8T 4K6, Canada A company Safe Software was formed to provide tools and training for the SAIF data standard. They may be contacted at `infosafe@safe.com` or by phone at either (604) 241-4424 or (604) 583-2016.

3 Assessment of some Code-level Data Access Tools

Code-level data access tools are concerned with what might be called the micro-management of data, in particular with handling the mapping of abstract data relationships to Fortran or C data strucutures (and hence computer memory). Since, however, they also provide an interface between application programs and data storage, they are also often seen by application programmers as a data storage solution by defining the structure and organisation of data files, and as a means of sharing data between applications.

Two such tools, now becoming widely used, are HDF and netCDF. They share many of the same goals: portability of data, self-description and compactness, but achieve these goals in slightly different ways. To gain experience with these tools we defined a pair of basic tasks: to write a data file from one program; and re-read it from another. The data we chose to use is a description of a Finite Element mesh which was itself read in from a neutral file format defined in the 1980s by the RALBIC project and used in the EU-funded EVEREST project [1]. EVEREST was one of the world first fully three-dimensional semiconductor device simulation packages. The software will simulate the electrical behaviour of three dimensional semiconductor devices under steady state and time dependent conditions by solving the classial drift-diffusion equations for the holes and electrons moving within a silicon based semiconductor.

In order to set the context in which we approached the use of HDF and netCDF for this data, the RALBIC neutral file system and a database, Deva, which uses the ISO standard STEP for its data modelling and Applications Programming Interface (API), are reviewed in Sections 3.1 and 3.2.

[1] `http://www.cse.clrc.ac.uk/Activity/EVEREST`

3.1 RALBIC – A Neutral File Format for Finite Element Data

3.1.1 Background

RALBIC is a well-defined neutral file format for the transfer of Finite Element (and Finite Volume) data. It is supported by a library of read and write routines written in FORTRAN77, and these provide access to commonly used data aggregates on irregular meshes, such as nodal coordinates, element topologies, nodal solutions, etc.

The RALBIC neutral files were originally designed to facilitate the exchange of data between software for electromagnetic modelling using Finite Element techniques. They were later extended to cover semiconductor device modelling, still using a Finite Element description of the discretisation scheme.

3.1.2 Information Storage

A RALBIC neutral file is a simple ASCII text file (though a binary version is also available). Within the file, data is grouped into sections delimited by $XXXX and $END-XXXX (where XXXX is a unique four-letter identifier for the section. Some sections can be nested. The format of each section is tightly specified, but the application programmer does not need to know the details as each section is written and read by a specific library routine. Thus, for example, to read the $NODE section, the routine RDNODE is used, to write the Dirichlet nodal conditions, WRDIRI is used.

3.1.3 Data Modelling

The data model used by RALBIC is highly domain specific (though parts of it can and have been re-used in other domains). In addition, certain assumptions are made about the data-structures, which will be used to represent the data within the application program, and data is returned from the API routines into these structures. If, for some reason, the program uses other structures, then an overhead is paid in assigning workspace and performing translations between data structures (e.g. transposing arrays). This illustrates the challenges faced.

3.1.4 API

As said above, the API is a library of routines, written in FORTRAN77, which read and write sections of the neutral file. In addition, there are some utility routines to open, close and initialise files and to nest sections. Because it is not possible to know the order in which sections have been written, RDGRPS looks ahead to identify the type of the next section. Then the program can either read the section using the apropriate routine or use RDSKIP to move on to the following section.

The data in this file is as follows (exclamation marks delimit comments):

```
$HEAD                   ! Identifies file as RALBIC file. Sections are
VER_03_87               ! delimited by $XXXX. A $HEAD section encloses
$CASE                   ! the whole file, $CASE sections separate out
MESH                    ! logically connected parts of the data.
$NODE
   68    3              ! number of nodes and dimensionality
```

```
    1  0.0000000D+00  0.0000000D+00  0.0000000D+00   ! Node number, x, y, z
    2  0.0000000D+00  0.0000000D+00  0.1000000D-03   ! coordinates
    3  0.0000000D+00  0.1000000D-03  0.1000000D-03
    :
    :
   68  0.9375000D-03  0.1000000D-03  0.1000000D-03
$END-NODE
$NTYP                ! Nodal type section - used to define boundary nodes
   68                ! number of nodes again (note redundancy)
    1   -1    2   -1    3   -1    4   -1    5    1    6    1    7    1    8    1
    9   -2   10   -2   11   -2   12   -2   13    0   14    0   15    0   16    0
    :         ! format is node number node type, node number, node type ...
    :
   65    0   66    0   67    0   68    0
$END-NTYP
$ELEM         ! Element information section
   16    8    ! Number of elements, maximum number of nodes per element
    1 BR08        SILICON    8    1   13   20    4    2   27   34    3
    2 BR08        SILICON    8   13   14   21   20   27   28   35   34
    3 BR08        SILICON    8   14   15   22   21   28   29   36   35
    :         ! element number, element type, element material, list of nodes
    :
   16 BR08        SILICON    8   47    9   12   54   61   10   11   68
$END-ELEM
$END-CASE
$END-HEAD
```

3.2 Deva – a STEP compliant database

3.2.1 Background

Deva is a database system which is based on the ISO 10303 STEP standard model of data description [9]. The data model can be defined through EXPRESS, the STEP data modelling language, and two APIs are available, a set of native Deva routines and a STEP Data Application Interface (SDAI) layer.

3.2.2 Information Storage

A Deva file is a binary representation of the data and its associated data model. STEP data is organised into *repositories* (in Deva a repository is a UNIX directory) and within a repository into *models* (an unfortunately confusing nomenclature). The data can be navigated through its entity-attribute relationships.

3.2.3 Data modelling

The simple data model which the RALBIC file format encapsulates can easily be written in EXPRESS, the STEP data modelling language. In fact there are a number of ways in which this can be done. A simple one is as follows:

```
SCHEMA fel;

  TYPE element_type = ENUMERATION OF
```

```
    (TE04,
     BR08);
  END_TYPE; -- element_type

  TYPE material = ENUMERATION OF
    (SILICON,
     OXIDE,
     AIR);
   END_TYPE; -- material

   ENTITY nodes_description;
      totnod : INTEGER;
      dimen  : INTEGER;
      nodes  : ARRAY [1:?] OF node;
      ntyp   : ARRAY [1:?] OF INTEGER;
   END_ENTITY; -- nodes_description

   ENTITY node;
      node_number : INTEGER;
      coord       : ARRAY [1:3] OF REAL;
   END_ENTITY; -- node

   ENTITY element;
      element_number  : INTEGER;
      el_type         : element_type;
      material        : material;
      number_of_nodes : INTEGER;
      nodes           : ARRAY [1:?] OF INTEGER;
   END_ENTITY; -- element

   ENTITY elements_description;
      totels               : INTEGER;
      max_nodes_per_element : INTEGER;
      elements             : ARRAY [1:?] OF element;
   END_ENTITY; -- elements_description

END_SCHEMA; -- fel
```

There are known deficiencies with this data model for practical applications, although it is closely related to the STEP standard for Finite Element models. One major problem is the presence of implicit data such as the need to interpret the triple of numbers forming the `coord` array for a node as Cartesian (x, y, z) rather than, say, polar (r, θ, ϕ) and the ordering of nodes within an element.

Another problem for efficient access is the granularity of the data. Since, for a strictly STEP compliant interface, only individual attributes of entities, and only individual elements of array attributes, can be accessed at a time, the number of subroutine calls to read or write this model would be huge for any decently sized mesh. We have overcome this in the past by extending the SDAI to allow the reading and writing of whole arrays with one procedure call. This has been implemented on an SDAI layer over Deva and was shown to give good performance.

3.2.4 API

There are two APIs to Deva, a native one and the STEP defined SDAI. Both of these are in C, but in the EU-funded TOOLSHED project [2], we wrote an unofficial FORTRAN77 binding to the SDAI which we have also used with Deva.

Use of Deva starts by opening or creating a repository, then opening or creating a model within the repository. From there on the data model is traversed – creating or accessing an entity (actually an instance of an entity since there can be several within a model) returns an identifier which can be used to access attributes. Basic attributes may be strings or numbers and read as such, others may themselves be identifiers of entities within the database.

3.3 HDF – Hierarchical Data Format

3.3.1 Background

HDF is a file format and an associated applications interface library, see Section 2.9. The current version is HDF5 which is considerably changed since HDF4.x. In HDF there are four basic entities: the file, the group, the data set and the attribute.

3.3.2 Information Storage

An HDF file is a container in which data is stored. It is usually (though not necessarily) mapped onto a file in the operating system's directory structure. Within the file, data is clustered into groups (c.f. the $CASE section of the RALBIC neutral file described in Section 3.1). Thus, for example all the data associated with one output time of a transient computation could be put into one group.

3.3.3 Data Modelling

Inside each group the data is arranged in data sets which are multi-dimensional arrays of data elements with the meta data that describes them (number of dimensions, size of array, data type, etc.). Attributes can be applied to any group or data set and are user defined structures to contain aditional information about an object. For example, a text string containing the units to be used when interpreting a data item could be stored in an attribute.

Groups and data sets are arranged in an hierarchical tree structure, much like a UNIX directory tree. In fact, HDF even uses a UNIX-like syntax (`/group/subgroup/dataset`) to navigate this tree. In this sense, groups are analogous to directories and data sets to files. A special group which is automatically present in any HDF file, is the "root" group designated by "/". There is no facility to express any relationships between data other than this tree-like structure.

3.3.4 API

There are bindings for HDF for C, Fortran 90 and Java. It is possible that the Fortran 90 interface would also support FORTRAN77 codes, though it is likely that they would need to be compiled with a Fortran 90 compiler and some functionality would be lost.

[2] `http://www.cse.clrc.ac.uk/Activity/TOOLSHED`

An HDF session starts with creating a new file or opening an exisiting one (in fact, in the Fortran 90 version, the first call is to a routine which initialises some datatypes). Likewise, finishing a session ends with closing the file which finishes updating the data held on disk and cleans up all memory which the library has used. Again, in Fortran 90, it is necessary to follow this with a call to close the defined datatypes. After the file has been opened for writing, a group can be created. As the simple mesh data we wished to store did not require it, we skipped this step and so our data sets were created in the root group.

To create a data set it is first necessary to create a data space, which defines the dimensionality and size of the array (recall that all data sets are either scalars or multi-dimensional arrays). To do this it is sufficient to give HDF the type (integer, real, double, etc.), the rank (0 for scalar, 1 for vector, etc) and the ranges of the dimensions. HDF returns an identifier which can then be used to create the data sets referencing that data space. The data set can then be written with a simple call. Since we were writing in Fortran 90, we were able to use the array section notation to ensure that only the relevant data was written. For example, consider the element topology array. In many applications this will be max_elements by max_nodes_per_element, but only number_of_elements by nodes_per_element will be used. So if the code allows for 10000 elements with up to 20 nodes each, but the mesh is only 16 brick elements with 8 nodes apiece, it would be excessive to write out all 200000 data locations. Array sectioning allows us to only pass `eltop` (1:16,1:8) to the HDF subroutine where it is treated as contiguous storage (see example below).

Reading is a simple reversal of the process if the names, sizes and types of the data are known. Each data set is written with an associated name and this is used to refer to it when opening the data set for reading. This method of access requires an agreement to be reached in advance on the names and structure in the data model, which conflicts somewhat with HDF's aim of being self describing. To overcome this there are a number of inquiry functions which allow a program to iterate over all data sets in a group and discover their type, extent and names. As is so often the case in data exchange, this ability does nothing to solve the problem of interpreting unknown data, merely pushing it down one more level. As a concrete example, the program may be able to find out that one of the data sets in the file is called `coord`, say, that it is a three by something double precision array, that so many of its elements have been given a value, but the knowledge that the coordinate array represents the Cartesian positions (x, y, z) of a set of Finite Element nodes rather than a vector-valued solution function defined on those nodes, has to be agreed upon outside HDF.

3.4 NetCDF – Network Common Data Format

3.4.1 Background

NetCDF shares many characteristics with HDF, in particular its orientation towards array data. See Section 2.3.

3.4.2 Information Storage

NetCDF data is held in a binary file with a well-defined (and published) format. Because this format is known it is possible to write software tools to manipulate, interrogate and otherwise make use of the data. Several such tools are included with the distribution.

3.4.3 Data Modelling

In addition to a specified file format and application program interface, netCDF has a data definition language, CDL (Common Data format Language) which, like EXPRESS, allows a formal description of the data model to be used. CDL is, however, considerably less powerful (one might say *less expressive*) than EXPRESS, reflecting the emphasis on the representation of data rather than the relationships between data "atoms".

CDL does have the advantage that it can be compiled directly into either a netCDF file or a program in C or FORTRAN77 which will define and write the data specified. Conversion from a netCDF file to CDL is also possible.

We used CDL to define our data model for netCDF and the `ncgen` utility provided with the distribution to generate FORTRAN code to create, define and write the data. This code was then used as an exemplar to interface to the RALBIC neutral file routines, see Section 3.1. The CDL definition (the `data` section contains dummy placeholders) was:

```
netcdf fel {
dimensions:
Dr = UNLIMITED ;
dimen = 3 ;
ieltop = 10 ;
imat = 10;
variables:
int totnod;
int totels;
double coord(Dr,dimen);
coord:units="cm";
int eltop(Dr, ieltop);
int ndtype(Dr);
char material(Dr,imat);

data:
totnod = 4;
totels = 1;
coord = 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0;
eltop = 7, 1, 1, 2, 3, 4;
ndtype = 1, 1, 0, 0;
material= "SILICON", "OXIDE", "AIR";
}
```

As can be seen the CDL form is more succint than EXPRESS, but it fails to define relationships between entities, such as that the `eltop` array anove is an array of nodes (in fact this eltop has its first two elements reserved for the material and the element type, coded as integers – to that extent this is far more a formalised description of the FORTRAN datastructures used by the code).

On the other hand the FORTRAN source generated by `ncgen` is verbose and difficult to read. For example, to define the `coord` array, the code defines the variables `coord_id, coord_rank` (a parameter), `coord_dims(coord_rank), coord_start(coord_rank), coord_count(coord_rank)` and finally `coord` itself. Similarly for the other arrays. In a large data model, this could mean an enormous number of variables and a high chance of names similar enough to cause confusion. It is important for the hardened FORTRAN programmer to realise that in the CDL, array subscripts are given in C ordering and converted to FORTRAN ordering (first runs fastest) by `ncgen`.

3.4.4 API

In addition to C and FORTRAN77, C++ is officially supported and there are contributed APIs in Perl, matlab-5, Python and Java. Fortran 90 is on Unidata's development plan.

NetCDF sessions have two distinct phases – one where the data is described when netCDF is in *define mode* and the other where data is written or read in *data mode*. It is possible to switch between modes (for example to re-define the bounds of an array), but normally speaking *define mode* is called once when the netCDF file (confusingly called a dataset) is created, then left for *data mode*.

There are basic entities in netCDF: variables, which in general will be arrays; attributes, which house metadata for variables; and dimensions, which determine the bounds of arrays. All three can be referred to by name or by referencing a unique identifier which netCDF will return. As it is expected that many bounds of arrays will be related these are expressed as separate from a variable's attributes – for example, in our data model the nodal coordinate array will be $3 \times$ maxnod and the nodal type maxnod long. In *define mode*, dimensions can be defined (given a name and a value which is limited to being a determined integer; it is not possible to provide an expression such as $5 \times$ maxnod), define variables (dimensionality, size, type) and assign attributes. In contrast, in *data mode* the main actions are simply to write variables and to read variables, dimensions and attributes, though various enquiry functions are also available.

One dimension is allowed to be defined as unlimited and is not given a specific value in the *define mode* phase. Instead it acquires its value as arrays are written using it, retaining the largest value used.

Since the API we used for the tests is written for FORTRAN77, the array sectioning trick we used with HDF (Section 3.3) was not available to deal with arrays declared of different size to that expected. Instead, netCDF supplies several different routines for writing and reading arrays from element by element access to whole array access to mapped arrays where start points and strides are specified. This last can mimic the array section functionality of Fortran 90.

One final issue that gave a little difficulty is the matter of CHARACTER arrays. In FORTRAN it is permissible to define an array `CHARACTER*(10) MAT(4)`. The netCDF char type is equivalent to `CHARACTER*(1)`, so MAT would have to be declared in the CDL as a two dimensional array, `char mat(4,10)` – this is like `CHARACTER*(1) MAT(10,4)`. Fortran is in any case not designed for handling character strings.

4 Conclusions

We have compared the use of four code-level data management tools in the relatively simple task of reading and writing a finite element mesh. Two of the tools were in-house libraries representing two extreme ends of the current approaches to data-management. RALBIC is a neutral file system with a rigidly-defined structure and the only insulation the programmer has from that structure is through an API with essentially only one routine to read and one to write each of the permitted sections of the file (and hence each of the data entities). Extension of the data model is possible by overloading data sections or by writing additional routines. In contrast Deva is a data repository where both the representation of the data as stored and the access to it are driven by a formal data model written in EXPRESS.

The other two systems studied, HDF and netCDF, are publicly available and fall between these two extremes. netCDF has a data modelling language, though it is nowhere near as rich as EXPRESS and the data representation is consequently limited (though it is adequate for many scientific applications). The hierarchical nature of HDF is appealing, and its Fortran 90 interface with the use of array sectioning to make efficient read/write operations is a major advantage. The API is fairly straightforward to understand and implement – the same cannot be said of netCDF, but here the use of `ncgen` to generate example code was very useful. It will be even more useful when the Fortran 90 binding becomes available.

References

[1] K. Kleese *Requirements for a Data Management Infrastructure to support UK High-End Computing* Technical Report DL-TR-99-04 (Daresbury Laboratory, 1999)

[2] K. Kleese and R.J. Allan *Proc. DM conf.* (Daresbury Laboratory, 2000)

[3] K.P. Duffey and C.R.I. Emson *Ralbic – A Simple Neutral File for Finite Element Data* Technical Report RAL-87-103 (Rutherford Appleton Laboratory, 1987)

[4] D. Thomas and C. Greenough *The MIDAS SDAI Implementation* Rechnical Report RAL-TR-96-043 (Rutherford Appleton Laboratory, 1996)

[5] R.J. Allan *Survey of Computational Grid, Meta-computing and Network Information Tools* Edition 2. Technical Report DL-TR-99-01 (Daresbury Laboratory, 2000) See `http://www.dl.ac.uk/TCSC/HPCI/reports.html`

[6] R.J. Allan, J.M. Brooke and F. Costen and M. Westhead *Grid-based High Performance Computing* Technical Report of the UKHEC Collaboration (UKHEC, 2000) See `http://www.ukhec.ac.uk/publications`

[7] R.J. Allan, J.M. Brooke and F. Costen and M. Westhead *The UK HEC Grid: state of the art and next steps* Technical Report of the UKHEC Collaboration (UKHEC, 2000) See `http://www.ukhec.ac.uk/publications`

[8] C. Greenough, R.F. Fowler and R.J. Allan *Parallel IO for High Performance Computing* (UKHEC 2001)

[9] ISO 10303 STEP: Standard for the Exchange of Product Model Data. See `http://www.steptools.com/library/standard/introduction_to_step.html`

[10] The VICAR Users' Guide is available from `http://rushmore.jpl.nasa.gov/PAG/public/vug`