GGF DFDL Use Cases

Status of This Memo

Copyright Notice

Abstract

Revision History

Latest entry ***at the top*** please			
Version	Author/Contributor	History	Date(yyyy-mm-dd)
	Robert E. McGrath		2005-09-23

Contents

Abstract 1
Revision History 2
1. Introduction 4
2. Attribute-Relation File Format (ARFF) 4
3. Relational Database 5
4. Grid Application 9
5. Example: MODIS Level 2 Data (NASA Earth Observing System) 12
 5.1 The Data Product 12
 5.2 Views of the Data 13
 5.3 The File Layout 14
 5.4 DFDL Description(s) for this Dataset 17
 5.5 Layers 17
6. References 17

1. Introduction

This note presents some possible use cases for the Data Format Definition Language (DFDL).

These use cases are intended to instigate discussion and guide design. These cases do not reflect formal requirements on the DFDL.

2. Attribute-Relation File Format (ARFF)

An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files were developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato for use with the Weka machine learning software ([7]).

This format is similar to many other simple text formats. There is a header with free text, followed by data definitions that define one dimensional records, followed by a sequence of records in comma delimited strings.

```
% 1. Title: Iris Plants Database
 %
 % 2. Sources:
 %      (a) Creator: R.A. Fisher
 %      (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
 %      © Date: July, 1988
 %
 @RELATION iris
 @ATTRIBUTE sepallength  NUMERIC
 @ATTRIBUTE sepalwidth   NUMERIC
 @ATTRIBUTE petallength  NUMERIC
 @ATTRIBUTE petalwidth   NUMERIC
 @ATTRIBUTE class        {Iris-setosa,Iris-versicolor,Iris-virginica}
 @DATA
 5.1,3.5,1.4,0.2,Iris-setosa
 4.9,3.0,1.4,0.2,Iris-setosa
 4.7,3.2,1.3,0.2,Iris-setosa
 4.6,3.1,1.5,0.2,Iris-setosa
 5.0,3.6,1.4,0.2,Iris-setosa
 5.4,3.9,1.7,0.4,Iris-setosa
 4.6,3.4,1.4,0.3,Iris-setosa
 5.0,3.4,1.5,0.2,Iris-setosa
 4.4,2.9,1.4,0.2,Iris-setosa
 4.9,3.1,1.5,0.1,Iris-setosa
```

Figure 1. Adapted from [cite].

The DFDL use case has two variants:

1. ingesting ARFF data
2. converting data into ARFF

Figure 2 is a sketch of the first case: an application needs data in a specific format, but needs to read data from ARFF. A DFDL-aware reader will be able to use a DFDL description to read the ARFF data into the required internal layout.

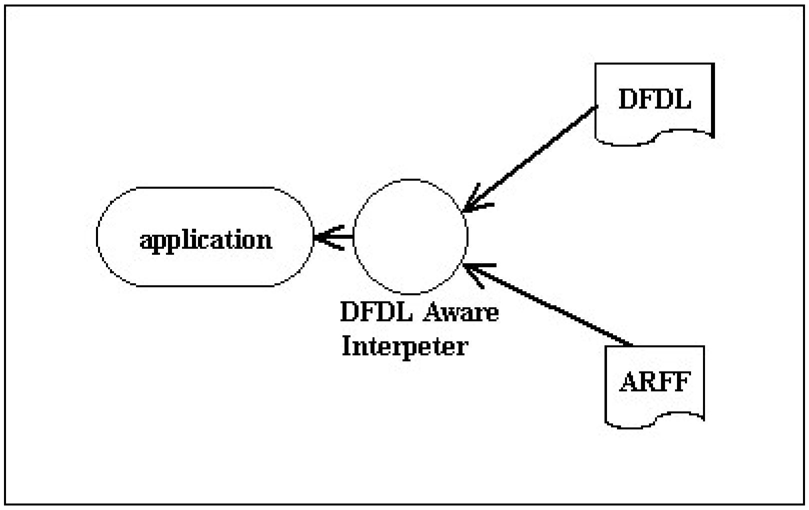


Figure 2

This case is interesting because the ARFF itself includes a data definition language. This means that the DFDL should actually be generated for each ARFF file, i.e., the declaration part of the ARFF should be used to generate a description of the records.

Figure 3 is a sketch of the second case: the program needs data in ARFF, but needs to read data from other sources. In this case, the DFDL is used to parse the data, which is written in a conventional ARFF file.

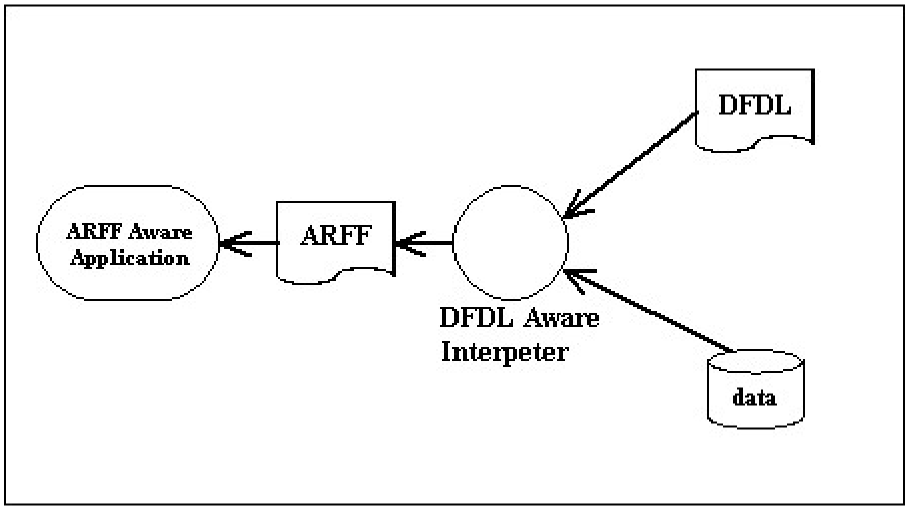


Figure 3

Again, this case is somewhat interesting, because the ARFF needs to have a data definition at the front. The DFDL-aware reader will use the DFDL to generate this header information in the output ARFF.

3. Relational Database

Many Grid applications will be accessing relational databases through custom or standard interfaces. This will be analogous to conventional client-server interactions:

1. The application (client) will issue a query in a structured language

2. the data server (database) will process the query, to formulate an answer
3. the result will be returned to the client in a standard format, e.g., as a table.

Figure 4 shows a sketch of this operation.

The client and database share a data definition language and a common schema for the data in question. These are used to formulate a valid query, to interpret query, to formulate a valid response, and to interpret the response (Figure 5).

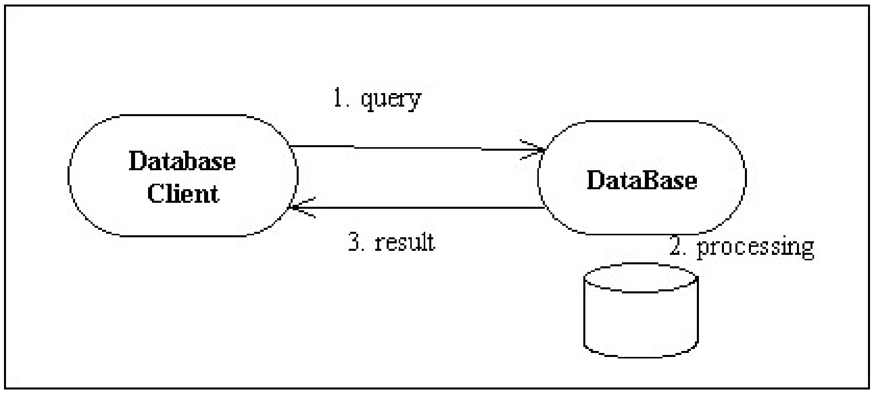


Figure 4

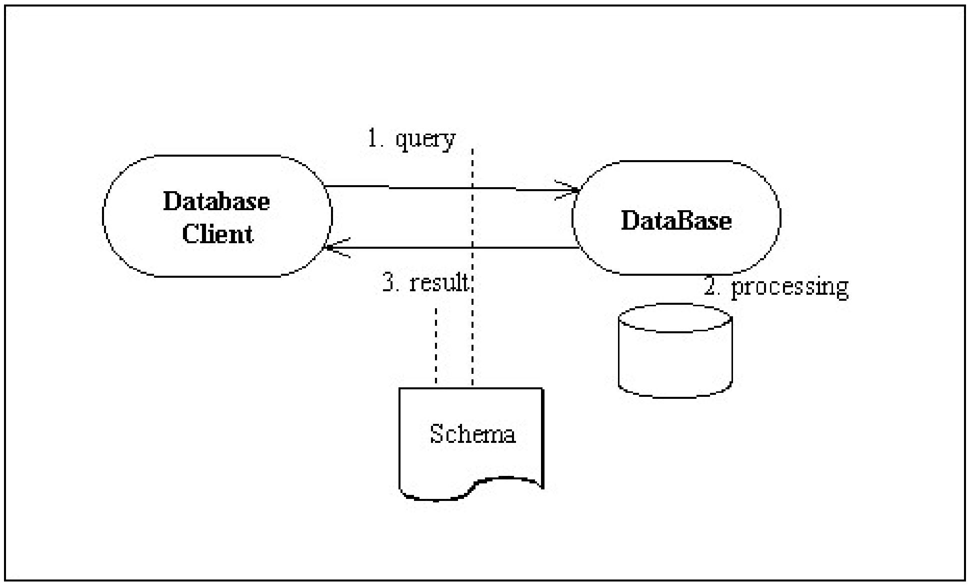


Figure 5

In a Grid environment, data users and data sources will be less tightly coupled, and applications will want to use data from many sources which may have many different schemas. The DFDL can enhance interoperability by providing a standard way to define the data, independent of the specific database language and schema.[1]

Figure 6 sketches a possible use of DFDL to define the format of the query and answer. The general idea is that the DFDL would enable a generic query to be translated into specific query format for one or more physical databases. The DFDL description of the answer will enable the receiver to decode results from different sources.

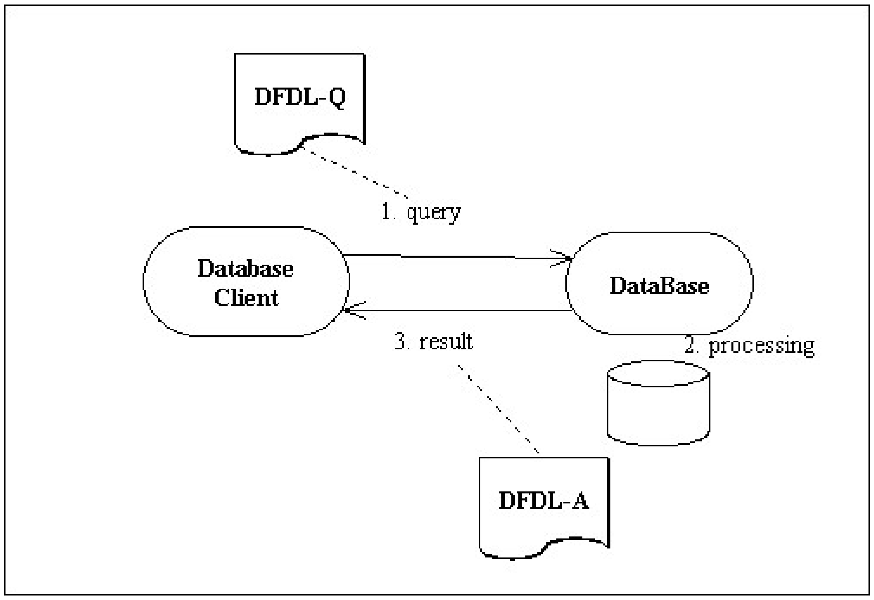


Figure 6

The database or database service presents data in a canonical form, e.g., as a table. This canonical form is what the DFDL in Figure 6 is describing.

However, the data service probably does not store a simple image of the table described by the schema. Instead, the data may be stored in paged memory, with one or more indexes, and multi-level caches. Some parts of the table may be computed instead of store. Thus, the actual stored representation of the data may be quite different from the view shown at the interface (Figure 7). The relationship between the stored data and the view is quite complex and depends on the implementation.

In this more extensive view, the DFDL could be used in several addition parts of the system. The structure of the database at the interface could be described (Figure 8, DFDL-T). For example, a database table could be described just as if it were a flat file.

Inside the server itself, the data structures that make up the table could, in principle, be described by DFDL layers (Figure 8, DFDL-Layers).

[1] Automatically translating between database schemas is very difficult (believed to be undecidable). But once such mappings are created, the DFDL can be used to apply them automatically.

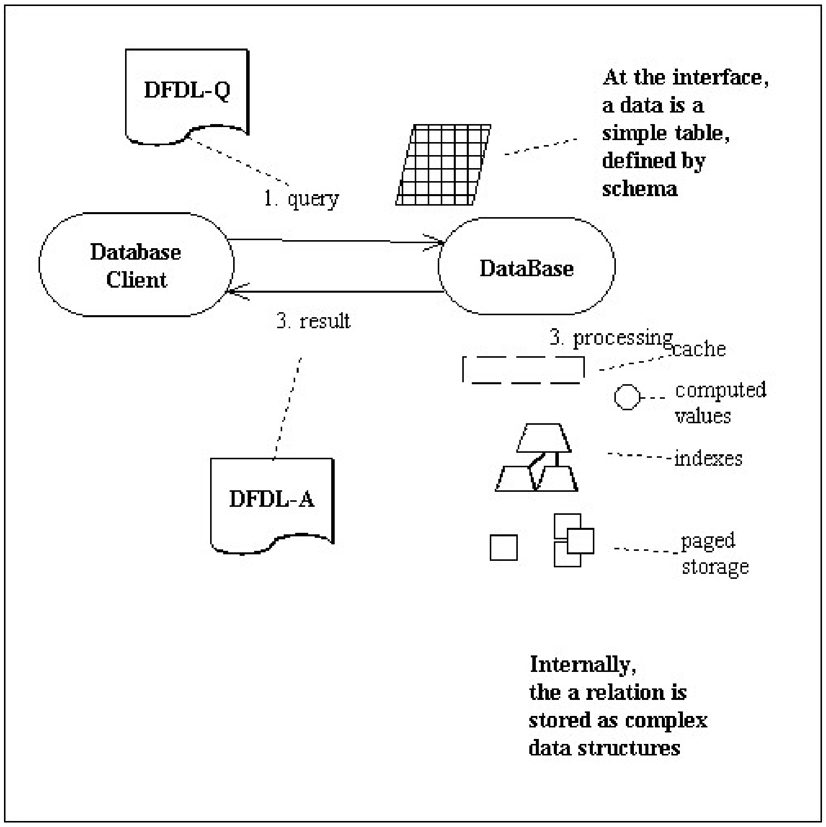

DFDL-Q
At the interface, a data is a simple table, defined by schema
1. query
Database Client
DataBase
3. result
3. processing
cache
computed values
indexes
DFDL-A
paged storage
Internally, the a relation is stored as complex data structures

Figure 7

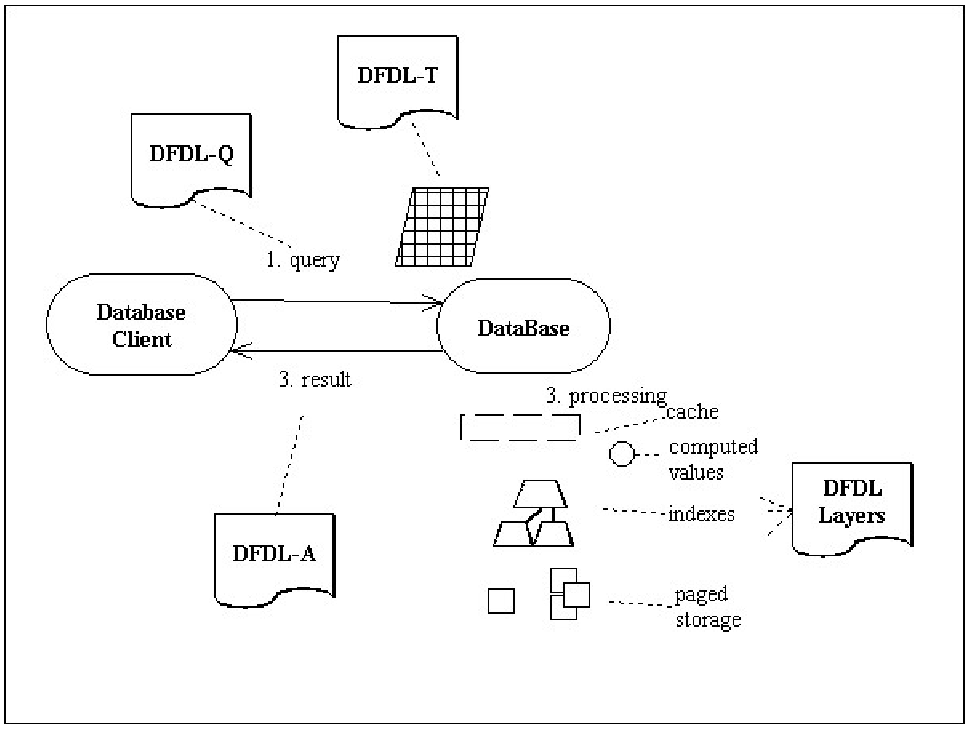


Figure 8

However, even if this DFDL were available, it is unlikely that it would be visible to the client. This illustrates a case where the data “format” of interest is not the physical bits.

4. Grid Application

This section is modeled after the Linked Environments for Atmospheric Discovery (LEAD) project, is a large, multi-institution effort to create an integrated, scalable cyberinfrastructure for mesoscale meteorology research and education [3].

Like the relational database case, it is a client-server model for data access, except the data may have no schema.

One important class of users will be remote access to data via Data Services. In this use case, the application accesses virtualized data, and should never need to see the stored objects. The services manage stored objects, and present data through their interfaces.

The implication of this virtualization is that there are two different uses of DFDL, one for the client/server interface, and the other for managing the physical data. It is important to point out that the client application definitely should *not* deal with DFDL descriptions of the physical data.

Consider a weather simulation running on the Grid. The program ingests initial conditions in the form of a large array, (latitude x longitude x altitude), containing values for physical parameters such as temperature, humidity, and wind velocity. (Figure 9)

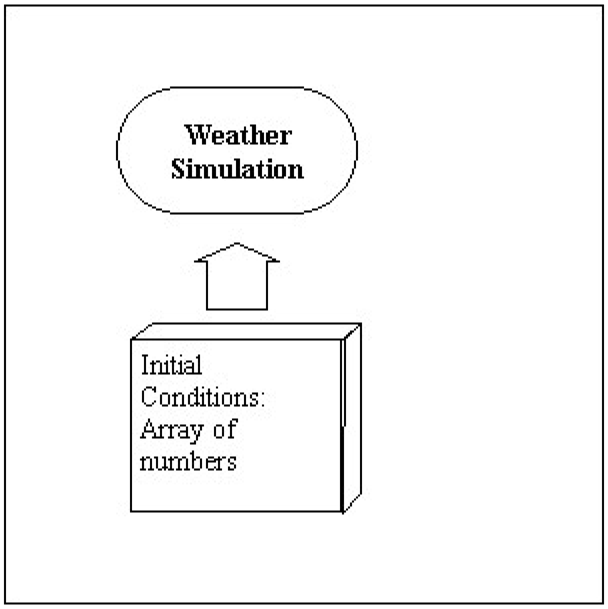


Figure 9

For the Grid application, this array of initial conditions should be virtualized, so that it can be populated by data sources from the Grid, such as satellites, radar, ground stations, and distributed archives. Ideally, the input array can be constructed by requests to Grid Data Services.

This would done through a “conversation with the data ([6])” suggested in Figure 10:

1. Send message to one or more services: “please give me data from (describe targets), in the following layout (describe the required input array)”
2. The service collects the data (if possible), and constructs a result.
3. The reply from the service may be “can’t do it”, or it might be “here is the data (which might not be the layout requested).

Note that DFDL can play a critical role in this exchange: it enables the parties to describe what they want and/or are really sending. Furthermore, it enables the client to work with data from many sources, so long as the service can provide the data needed along with a DFDL description.

Figure 11 suggests where DFDL descriptions would be used in this protocol. First, the input array could be described with DFDL. This description could also be sent as part of the data request message. The second use would be a description of the returned dataset. (This would likely include other metadata about the provenance and characteristics of the data.)

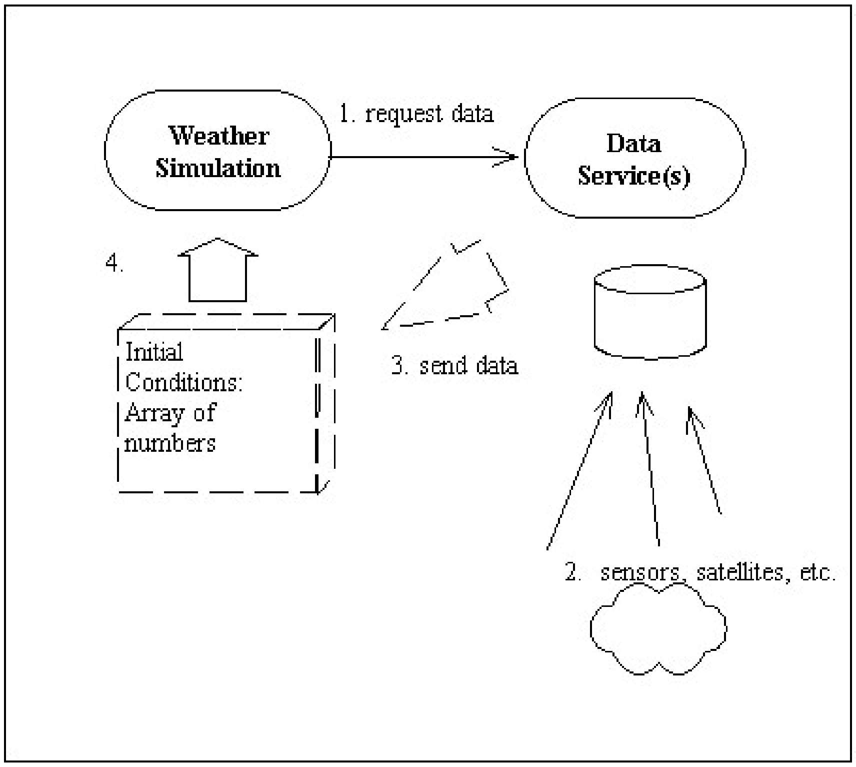

Weather Simulation
1. request data
Data Service(s)
4.
Initial Conditions: Array of numbers
3. send data
2. sensors, satellites, etc.

Figure 10

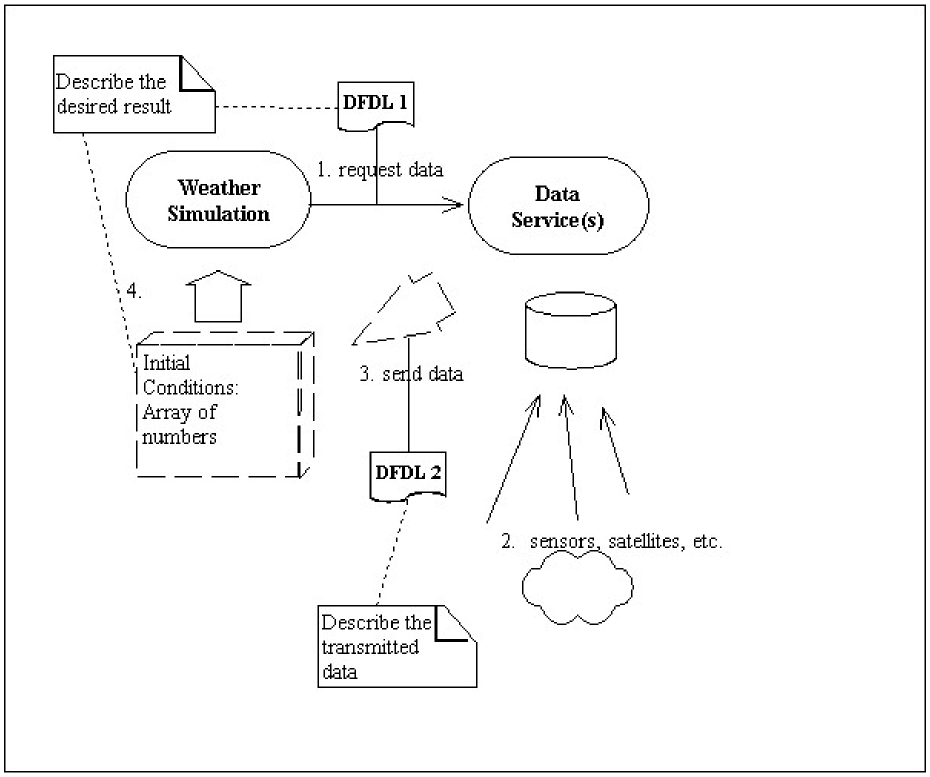

Describe the desired result
DFDL 1
1. request data
Weather Simulation
Data Service(s)
4.
Initial Conditions: Array of numbers
3. send data
DFDL 2
2. sensors, satellites, etc.
Describe the transmitted data

Figure 11

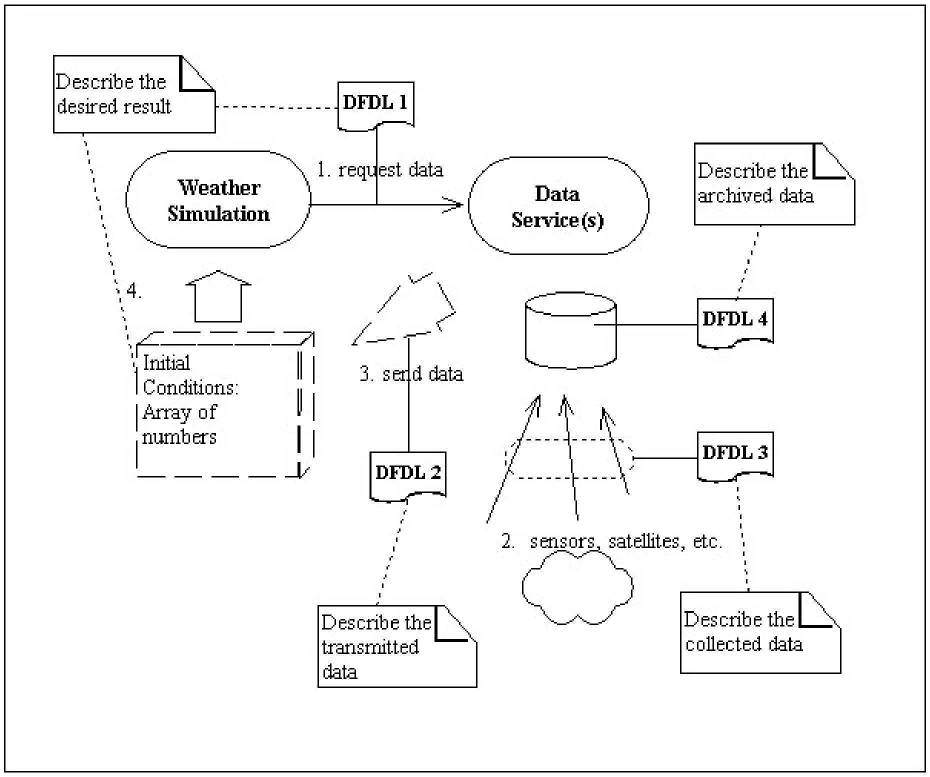
Both of these descriptions have no necessary relationship to the format of the stored data. As Figure 12 suggests, additional DFDL could be used on the service side, to describe the sensor data and the archived data.

The DFDL is useful in different ways on the two sides of the conversation.

For the application, it can use the DFDL to describe the format of its input files, either local or "virtual". This description can be used in requests to services.

The DFDL is a critical piece of the data service interface, defining the request and the response.

Within the data service, the DFDL is used to manage data from different sources, and to implement data transformation algorithms.



Describe the desired result
DFDL 1
1. request data
Weather Simulation
Data Service(s)
Describe the archived data
4.
DFDL 4
Initial Conditions: Array of numbers
3. send data
DFDL 3
DFDL 2
2. sensors, satellites, etc.
Describe the transmitted data
Describe the collected data

Figure 12

5. Example: MODIS Level 2 Data (NASA Earth Observing System)

The NASA Earth Observing System continuously measures the Earth, producing a Terabyte data per day in 2004. This data is distributed in a number of forms, and processed into hundreds of derivative products.

5.1 The Data Product

One of the key data products are the “Level 1B” data from the Moderate Resolution Imaging Spectrometer (MODIS), e.g., the 205m radiances (MOD02QKM) [1].

Each of these data sets contains 5 minutes of data, 288 datasets per day. The 250m data is 2 spectral bands, 1354 pixels per scan, 2030 scans per data product.[2]

The data is stored and distributed in HDF-EOS, which is a profile of the HDF4 binary format [2]. The data is organized into standard data products, each instance is a file with extensive metadata and an array of satellite measurements. The data is access via custom readers that are linked with the HDF-EOS and HDF libraries.

[2] Data, software, and documentation are available from NASA, e.g., http://daac.gsfc.nasa.gov/

5.2 Views of the Data

This dataset can be viewed in different ways. Physically, the data is collected as measurements from an orbiting satellite. The observations are horizontal scans across the orbital path of the satellite, which is a rectangle along the Earth (Figure 13a).

This data is stored in HDF-EOS as a single “swath” object [2]. The swath has multiple fields (one for each sensor band), with each pixel position geospatially located. That is, the latitude and longitude of each measurement is stored. Note that because of the geometry of the orbit, the data is not a rectangle with respect to the geo referenced grid (Figure 13b).

This object is stored as several HDF4 objects. The data is stored as a multidimensional array, with one plane per band. A second table holds the latitude and longitude for each position in the data array (Figure 13c). The file also contains substantial amounts of metadata as well.

The HDF objects are stored according to the HDF4 specification [3]. The file will contain a number of metadata objects, linked lists, and so on, which are managed by the internals of the library.

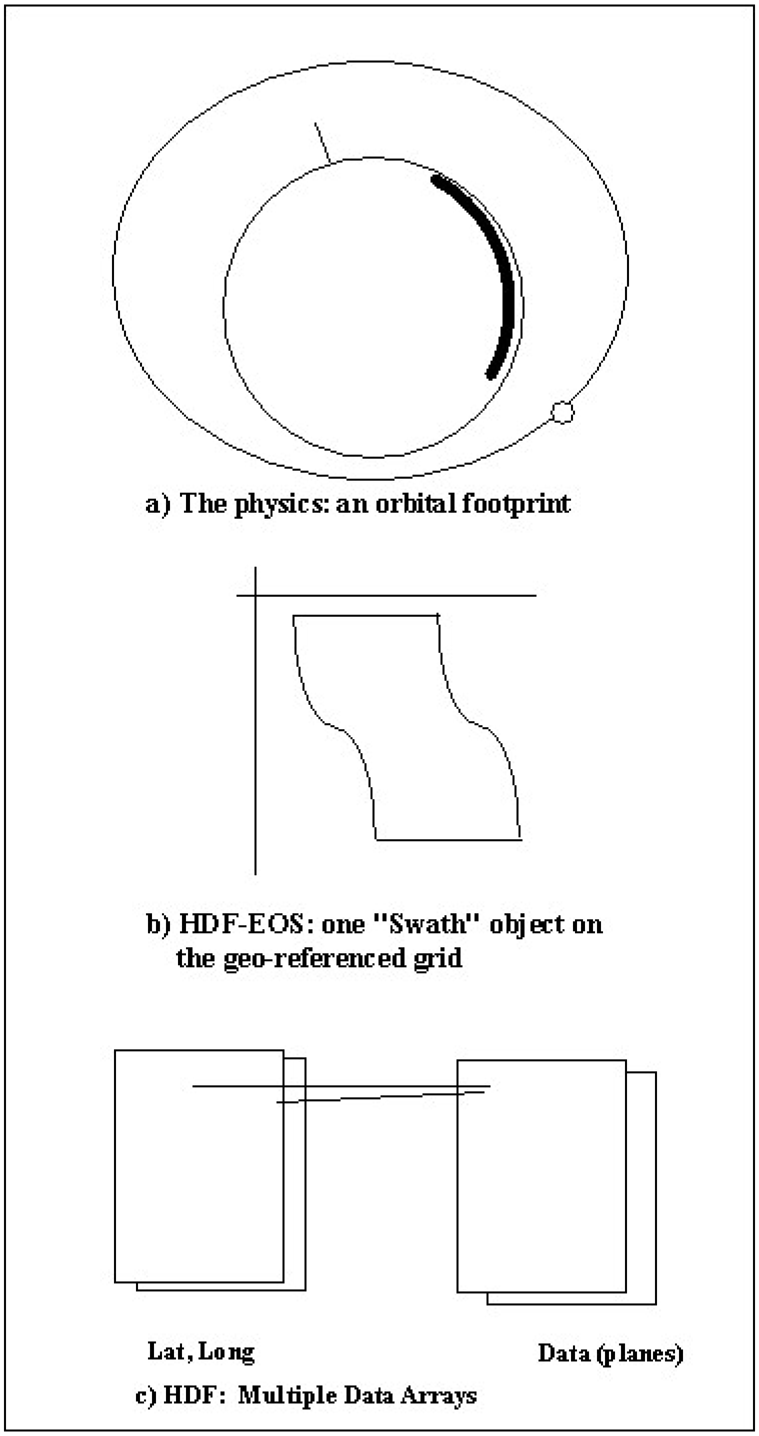

a) The physics: an orbital footprint
b) HDF-EOS: one "Swath" object on the geo-referenced grid
Lat, Long
Data (planes)
c) HDF: Multiple Data Arrays

Figure 13

5.3 The File Layout

Table 1 lists the objects in one of these data products, and Table 2 shows the attributes with metadata. Each file in this series will have the same objects.

Table 1. HDF4 objects in MOD02QKM.A2004317.0000.004.2004317072459.hdf (original size: 286,058,198).

Attr: global attributes, VG: VGroup, VD: Vdata table, SDS: Datasets.

Type	Object	Dimensions	Number type	Size (bytes)	Attrs
Attr	Global attributes (50)			76,474	
VG	`MODIS_SWATH_Type_L1B`				
VG	`MODIS_SWATH_Type_L1B/Geolocation Fields`				
SDS*	`MODIS_SWATH_Type_L1B/Geolocation Fields/Latitude`	2030x1345	Float 32	10,921,400	(3 attrs)
SDS *	`MODIS_SWATH_Type_L1B/Geolocation Fields/Longitude`	2030x1345	Float 32	10,921,400	“
VG	`MODIS_SWATH_Type_L1B/Data Fields`				
SDS *	`MODIS_SWATH_Type_L1B/Data Fields/EV_250_RefSB`	2 x 8120 x 5416	Uint 16	175,911,680	(14 attrs)
SDS *	`MODIS_SWATH_Type_L1B/Data Fields/EV_250_RefSB_Uncert_Indexes`	2x8120x5416	Uint 8	87,955,840	(7 attrs)
VD	`MODIS_SWATH_Type_L1B/Data Fields/Band_250M`	1		32	
VG	`MODIS_SWATH_Type_L1B/Swath Attributes`				
SDS*!	`Noise in Thermal Detectors`	16x10	Uint8	160	
SDS*!	`Change in relative responses`	16x10	Uint8	160	
SDS *	`DC Restore Change for Thermal Bands`	203x16x10	Uint8	32,480	
SDS *	`DC Restore Change for Reflective 250m Bands`	203x2x40	Int8	16,240	
SDS *	`DC Restore Change for Reflective 500m Bands`	203x5x20		20,300	
SDS *	`DC Restore Change for Reflective 1km Bands`	203x15x10		30,450	
VD	`Level 1B Swath Metadata`	203	64 bytes/ rec	12,992	

In this example product, there are dozens of objects, with three important blocks of metadata (StructMetadata.0, etc.), and a large payload of data (`EV_250_RefSB`) which is 2 x 8120 x 5416 elements, over 150MB.

These logical objects are written and physical records to the HDF4 file. The order and position of the objects in the file would depend on the precise sequence of writes in the program that creates the file.

Table 2. Global Attributes in MOD02QKM.A2004317.0000.004.2004317072459.hdf

HDFEOSVersion	11	char
StructMetadata.0	32,000	char
CoreMetadata.0	16,515	char
ArchiveMetadata.0	2,921	char
Number of Scans	1	Int32
Number of Day mode scans	1	Int32
Number of Night mode scans	1	Int32
Incomplete Scans	1	Int32
Max Earth View Frames	1	Int32
%Valid EV Observations	38	Int32
%Saturated EV Observations	38	Float32
% L1A EV All Scan Data are Missing	1	Float32
% L1A EV RSB DN Not in Day Mode	490	Float32
% L1A EV DN Missing Within Scan	490	Float32
% Dead Detector EV Data	490	Float32
% Sector Rotation EV Data	490	Float32
% Saturated EV Data	490	Float32
% TEB EV Data With Moon in SVP	490	Float32
% EV Data Where Cannot Compute BG DN	490	Float32
% RSB EV Data With dn** Below Scale	490	Float32
% EV Data Where Nadir Door Closed	490	Float32
% EV Data Not Calibrated	490	Float32
Bit QA Flags Last Value	1	Uint32
Bit QA Flags Change	1	Uint32
Granule Average QA Values	50	Float32
Electronics Redundancy Vector	2	Uint32
Electronics Configuration Change	2	Uint32
Reflective LUT Serial Number and Date of Last Change	21	char
Emissive LUT Serial Number and Date of Last Change	21	char
QA LUT Serial Number and Date of Last Change	21	char
Focal Plane Set Point State	1	Int8
Doors and Screens Configuration	1	Int8
Reflective Bands With Bad Data	22	Int8
Emissive Bands With Bad Data	16	Int8
Noise in Black Body Thermistors	12	Int8
Noise in Average BB Temperature	1	Uint8
Noise in LWIR FPA Temperature	1	Uint8
Noise in MWIR FPA Temperature	1	Uint8
Noise in Scan Mirror Thermistor #1	1	Uint8
Noise in Scan Mirror Thermistor #2	1	Uint8
Noise in Scan Mirror Thermistor Average	1	Uint8
Noise in Instrument Temperature	1	Uint8
Noise in Cavity Temperature	1	Uint8
Noise in Temperature of NIR FPA	1	Uint8
Noise in Temperature of Vis FPA	1	Uint8
Dead Detector List	490	Int8
Noisy Detector List	490	Int8
Detector Quality Flag	490	Uint8
Earth-Sun Distance	490	Float32
Solar Irradiance on RSB Detectors over pi	330	Float32

5.4 DFDL Description(s) for this Dataset

This dataset could be described by DFDL at any of these views.

From a scientific point of view, it would probably be most useful to have a DFDL description that describes the swath object: a single, array with geospatial coordinates (as in Figure 13b). This data is available through calls to the HDF-EOS library layer.

It would also be possible to describe the stored objects, i.e., the data arrays and georeference arrays (as in Figure 13c). This would describe several arrays, along with blocks of metadata. This could be accessed via the HDF library layer.

Finally, the layout on disk can (in principle) be stated precisely. It would be theoretically possible to write a DFDL description for a given MODIS file, i.e., the exact location, type, and size of the control and data blocks in the physical file. These would be accessible through standard file system calls.

Of course, the file level description would be both complex and distantly related to the conceptual view of the data as a single array with attached metadata.

5.5 Layers

The DFDL is designed to handle data transformations via “layers”. In this example, one could imagine modeling the multiple views with layers.

This would require multiple descriptions, and a mapping between them. Note that the mapping would be essentially a reengineering of the HDF and HDF-EOS library code (whose function is to implement this mapping).

More likely, the data should be described at an intermediate level (some form of array), with call outs to the appropriate library when the data is needed.

6. References

1. MODIS Characterization Support Team, “MODIS Level 1B Product User’s Guide”, MCSA Document # PUB-01-U-0202-REV B, NASA Goddard Space Flight Center, 2003.
2. http://hdfeos.gsfc.nasa.gov
3. http://hdf.ncsa.uiuc.edu/doc.html
4. lead.ou.edu
5. Attribute-Relation File Format (ARFF), http://www.cs.waikato.ac.nz/~ml/weka/arff.htm
6. Robert E. McGrath, Joe Futrelle, Ray Plante, Damien Guillaume, ``Digital Library Technology for Locating and Accessing Scientific Data'', accepted for *ACM DL'99*, August, 1999.
7. Ian H. Witten and Eibe Frank, *Data Mining: Practical Machine Learning Tools and Techniques (Second Edition)*, Morga Kauffnan, 2005.