18.11

FSL CENTRAL FACILITY DATA SYSTEM CONCEPTS

Paul Hamer[1]

NOAA Research - Forecast Systems Laboratory, Boulder, Colorado
[1]In collaboration with the Cooperative Institute for Research in the Atmosphere (CIRA), Colorado State University, Fort Collins, Colorado

1. INTRODUCTION

The mission of the Forecast Systems Laboratory (FSL) is to anticipate the science and technology that will be needed by the nation's operational atmospheric, oceanic, and hydrologic forecasting services in the next five to ten years. The Data Systems Group (DSG) of the Information and Technology Services (ITS) department within FSL is responsible for the acquisition, processing and distribution of some 150 GBytes of data a day in support of this mission. This ever-increasing volume coupled with the diverse formats used to describe these data are the main problems facing developers within DSG as they try to reduce the cost and increase the efficiency of the services provided. For a full description of the data and system configurations within the Central Facility see "Recent Advances in the FSL Central Facility Data Systems" (Lipschutz, 2005).

By the late 1990s the legacy systems, based on heterogeneous platforms running software developed within FSL using a functional decomposition design process, were becoming costly to maintain in both license fees for the various operating systems and, even more tellingly, software maintenance by the Data Systems Group. A decision was made at that time to invest in the effort of refactoring systems to achieve two major goals. The first was to switch to open source tools running under Linux and the second, to redesign and implement the data handling software using Object Oriented Analysis and Design (OOAD) techniques in order to reduce those software maintenance and operational costs.

The remainder of this paper concentrates on the second element of this effort, the software architecture changes and resulting implementation that became the Object Data System (ODS). It will include examples to illustrate these changes and how they have improved a CF function within FSL.

2. REQUIREMENTS

The data requirements placed on the CF are that for any requested dataset, it should provide timely access (near real-time if possible) to either the original dataset or some derived product, usually in the form of Network Common Data Form (NetCDF) (my.unidata.ucar.edu/content/software/netcdf). Further, the CF is to provide a retrospective access facility that supports requests to obtain original data or regenerated products for some limited period. The local archive is currently a mass store system that holds approximately four years of data. This retrospective element may also obtain datasets from other sources including the national archive at the National Climatic Data Center (NCDC), WSR-88D Level II data, for example.

*Corresponding author address: Paul Hamer, NOAA/FSL, R/FST, 325 Broadway, Boulder, CO 80305. Paul.Hamer@noaa.gov

3. SYSTEM DESIGN

To support these requirements, the following design choices were made:

- All data are to be held locally in the original receipt form;
- All data are to be labeled with a receipt or issue time;
- Translation of data is to be done only when necessary;
- Metadata are to be temporally consistent.

These four choices in themselves may not seem particularly significant, but they have led to an implementation that is flexible enough to support both real-time and retrospective processing within one system. Before discussing the implementation, let us first consider these choices in more detail.

- All data are to be held locally in the original receipt form. While software development is subject to a process that hopes to minimize the number of bugs, they are unlikely to be completely eliminated. Thus providing access to the original data format allows the reprocessing of these data if required following the fixing of a bug in metadata or software.

- All data are to be labeled with a receipt or issue time. Most data handled within the CF arrive in near real-time, and the ingest systems described below label these data with the receipt time. This is required because some formats do not include a full time and date element; rather they may include only the day of the month and not the month itself.

Therefore, in order to properly handle these data, we need to know either the receipt or issue time. Note that data obtained from archives outside of FSL must provide a means of obtaining the issue time to allow the correct handling of these data by ODS.

- Translation of data is to be done only when necessary. Together with the original format decision, we avoid unnecessary processing since much of the data are not used to derive products.

- Metadata are to be temporally consistent. When accessing metadata in support of data processing, we must be able to access the metadata valid at the receipt or issue time for those data. For example, a reporting station may be relocated and to support the retrospective processing of data provided by that station, we must be able to obtain the location information valid at that time.

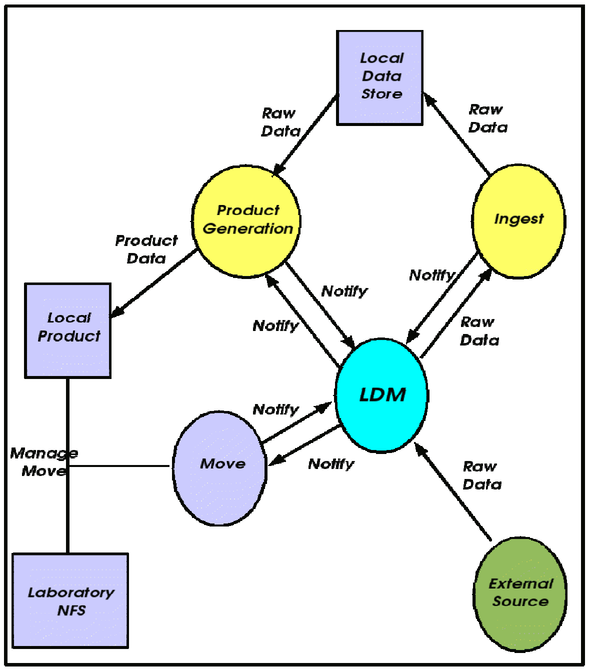

Figure 1: Data Ingest, Store and Notify Scheme

4. IMPLEMENTATION

The Central Facility ingests data from a number of sources using a variety of methods. Distribution of these data over the local network in a consistent fashion is key to a manageable system. For this task, DSG selected the Local Data Manager(LDM) (my.unidata.ucar.edu/content/software/ldm), a distributed system for event-driven data transport. LDM comprises a suite of software to select, capture, process, and distribute data products using a set of network client/server programs and their shared protocols. Available as a free distribution from Unidata that includes the package source code, this allowed DSG developers to extend the LDM to more fully support data handling by the Central Facility.

LDM allows distribution and access to data through the use of feed types and product keys. The product key was identified as a useful element to provide leverage within ODS; access to this element of the product description from the LDM was most easily obtained by mapping directly to the product queue. A description of how this was done will help illustrate the use of Design Patterns (Gamma, et al., 1995) within the design and implementation process. In this case the use of the Observer Pattern (see Fig. 2) was most appropriate as it defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. This is also known as publish-subscribe.

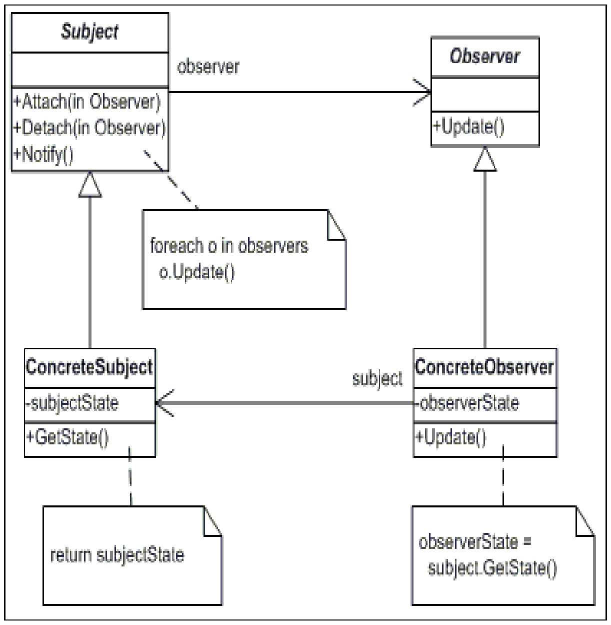

Figure 2: The Observer Pattern

The observer (LdmConsumer) and subject (LdmIngest) in this scenario are classes that encapsulate access to the LDM product queue. Consumers subscribe to the publisher using feed type and product key information for required products. The publisher will automatically update the consumer upon each update to the product queue that matches the subscription. The update consumers receive is another encapsulation, this time of the LDM data structure (LdmData) and it provides the consumer with details useful for efficiently dealing with large quantities of data. For example, one can discover how large a given message is without looking into the message contents. (As an aside, the particular implementation of this pattern within ODS allows for access to data from a LDM product queue without the LDM daemon actually running.) The ODS design then uses these LDM access abstractions to tier clients onto LDM, i.e. clients override access functions from the LdmConsumer in order to operate on data from the queue, such as in handling of World Meteorological Organization (WMO) GRIdded Binary (GRIB) (www.wmo.ch/web/www/WDM/Guides/Guide binary.html or (www.wmo.ch/web/www/DPS/grib 2.html) that is distributed over NOAAPORT (www.nws.noaa.gov/noaaport/html/noaaport.shtml).

The actual handling of GRIB leads to a discussion of two further concepts of ODS design: the generic handling of these data and the use of Virtual Inheritance to implement a form of Run Time Type Identification (RTTI) that allows an encapsulation of necessary function within a base class.

Throughout the ODS development process, an effort has been made to support the capture of data in such a way as to self document subsequent access to it, i.e. to associate a minimum set of metadata with the original data to support later analysis. Consider the ODS handling of GRIB: we store all GRIB messages (records) in a file that is named using the reference and forecast time. This file, in turn, is placed in a directory structure that is based on center, sub center, model and grid numbers. These numbers can be mapped to values in a lookup table that provide context. (e.g. 59/0/105/252 would map to FSL, Rapid Update Cycle (RUC) on a 20km Grid) The important thing to note is that the lookup values need not be known to store the raw data in a meaningful way.

To illustrate the utility of Virtual Inheritance, consider that while GRIB is being updated to Edition 2 of this format, this update will require some overlap between Edition 1 and 2. The use of Virtual Inheritance has enabled DSG to implement client software that is free from knowledge of the specific format. Thus the NetCDF making client for a GRIB dataset does not need to know what edition is being used, only that it is GRIB.

At this point it is worth mentioning that DSG considers meteorological data to be available in four distinct forms. These, with examples, are:

- Satellite
  GOES (www.oso.noaa.gov/goes)
- Radar
  Nexrad Level II (www.roc.noaa.gov)
- Point
  ACARS (web.usna.navy.mil/~bruninga/acar.html), BUFR (www.wmo.ch/web/www/WDM/Guides/BUFRCREX Guide-English.html), etc.
- Gridded
  Forecast model results

It is the handling of the Point data types that led to development of the OdsVar (or ODS variable) concept, linking named values and locations. For example, the name could be some well-known description, pressure say, with a value expressed as a float and valid unit (1001.3mb) at a location that consists of horizontal, vertical and temporal coordinates. All point data types can be reduced to a container of OdsVars when required. Thus, point data type translation will decode the original form into OdsVars. This generic object, coupled to the virtual inheritance RTTI allows for the development of clients, such as a NetCDF converter, without having to know anything of the original dataset.

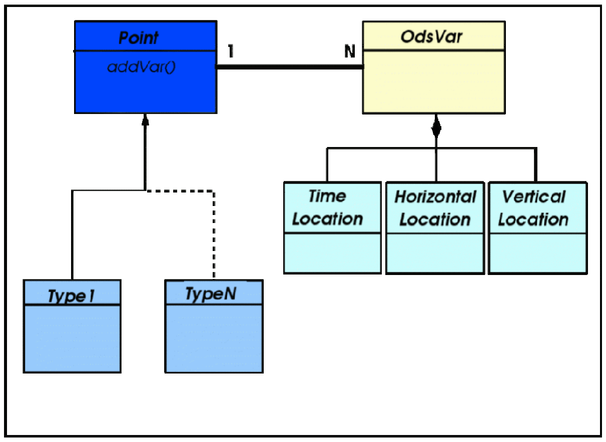

Figure 3: Point and ODS Variable Class Diagram

Of course, the use of RTTI and generic containers are only part of the picture for the complete handling of data within the CF. The final, and in some ways most important, part is the management of metadata.

5. METADATA

The question of metadata is one that could generate several additional papers for ODS in addition to this one, so its inclusion here is to illustrate specific implementation details.

Concentrating on the GRIB example used above, metadata in this context refers to the GRIB lookup tables that map center number, for example, to some name and description. In response to the design choice that all metadata must be temporally consistent, we currently store all table entries with a valid time field, namely an ISO 8601 (ftp://ftp.qsl.net/pub/g1smd/8601.pdf) time string that gives the UTC time at which this entry becomes valid. This requirement is the minimum required to support retrospective processing. A full bi-temporal database implementation (i.e. a history of histories) would be a valuable tool in support of ODS, or any system, but attempts to do so in-house have only been partially successful. It is hoped that an open source tool will soon be available so that we can resume work on this issue.

The selection of NetCDF for many of the derived products has allowed us to specify metadata as attributes of the required data. This in turn has reduced

software costs since NetCDF derived products from data forms already handled (GRIB, Point, etc.) require only that a CDL with appropriate attributes be created, not the writing of new software.

6. THE FUTURE

Much of the upcoming work in ODS development will concentrate on cataloging available data to assist in data discovery and support of ad hoc data requests. To do this, we are looking at using the eXtensible Markup Language (XML) (www.w3.org/XML) and schema such as Thematic Real time Environmental Distributed Data Services (THREDDS) (my.unidata.ucar.edu/content/projects/THREDDS) or the Earth Science Markup Language (ESML) (esml.itsc.uah.edu/index.jsp) both of which could help the CF support system interoperability.

Further, we hope to produce software tool sets to leverage work being done that maintains a bi temporal database for metadata. One interesting aside is that much of the meta data used to decode GRIB or BUFR are available as Portable Document Format (PDF) documents published at the WMO; these could more usefully be made available as XML.

In conclusion, the DSG at FSL has completed the refactoring of a large legacy data system using OOAD with Open Source and community tools. The result is one that satisfies FSL's CF data requirements and is "better, cheaper, faster" (Goldin, 1992).

References:

Lipschutz, R. C., C. MacDermaid, 2005: Recent Advances in the FSL Central Facility Data Systems

Gamma, E., R. Helm, R. Johnson, J. Vlissides, 1995: Design Patterns, Elements of Reusable Object-Oriented Software

Goldin, D., served as chief administrator of NASA for nine years, beginning in 1992. Using the motto, "better, cheaper, faster," he streamlined and restructured major NASA programs.