9.1 ANAGRAM—A MODULAR JAVA FRAMEWORK FOR HIGH-PERFORMANCE SCIENTIFIC DATA SERVERS

Joseph Wielgosz*
Center for Ocean-Land-Atmosphere Studies

1. SUMMARY

Anagram is a prototype framework designed to ease the development of the diverse data servers which will be needed as the community moves towards distributed data processing. It provides a collection of reusable components that address the needs common to high-performance scientific data servers. In particular, Anagram can greatly facilitate the development of new servers that support the OPeNDAP (a.k.a. DODS) subsetting protocol (Davis and Gallagher, 1999) on a diverse range of back-end data storage formats.

Developed at COLA (the Center for Ocean Land Atmosphere Studies, a center under the Institute for Global Environment and Society), Anagram is the basis for version 1.2 of the GDS (GrADS Data Server), which currently handles over 2 million hits per month at various sites (Wielgosz, Doty, and Adams, 2002)

This paper will explain the motivations for a common server framework, describe the functionality provided by Anagram's modular design based on Java, XML, and OPeNDAP (the Open Source Project for a Network Data Access Protocol), and explain how to extend and adapt it for specific applications, such as developing a new flavor of OPeNDAP server.

The Anagram home page is
http://www.iges.org/anagram/

2. BACKGROUND

2.1 Distributed data systems

Anagram is aimed at easing the current transition towards distributed systems for managing geosciences data.

At COLA and elsewhere, the quantities of data in storage are growing far faster than the capacities of the networks used to access them. As a result, it is becoming increasingly impractical to transfer datasets from one machine to another for analysis, comparison, and other kinds of reprocessing. Nevertheless, experience at COLA indicates that for many scientists, the ability to work with data in numeric form, using their own tools of choice, is indispensable.

Distributed data systems make this possible by allowing scientists to work directly with datasets stored remotely, either over a local network or over the Internet– for example, retrieving precise subsets instead of entire intact data files, or sending data-intensive analysis tasks to be run on a data server, with only the final results sent to the client. This is achieved by the adoption of common network protocols and usage conventions, which make it possible for diverse geoscience-oriented client tools and data servers, developed by different communities and institutions, to interoperate.

A distributed data system of some kind is now becoming a standard component of data-intensive research projects. This situation is leading to the development of a wide variety of data server software, to form bridges between the various protocol interfaces of distributed data networks, and site-specific or community-specific codes used to access and analyze particular data formats.

However, resources available for developing and maintaining infrastructure-related software are often very limited. A reusable data-server framework could help the community to make the best possible use of these resources by significantly reducing the development time required for the support of a new data format or protocol.

2.2 Data servers and web portals

It is worth emphasizing that the task of a data server, as discussed here, is distinct from that of a web portal system. The emphasis in a web portal system is on providing a feature-rich, human-oriented, user interface to one or more data archives, by generating dynamic web pages and images, and providing customized file downloads.

In contrast, a data server's main purpose is to support reliable machine-to-machine interactions, by exchanging encoded protocol requests and numeric data. The client may be a desktop application, an automated system driven by scripts, or another data server. For example, web portal systems are often primary "clients" for data servers, as in the case of the Live Access Server developed at the NOAA Pacific Marine Environmental Laboratory (PMEL), or the Ingrid system at Columbia University's International Research Institute for Climate Prediction - both of which provide a unified web-based interface to data from numerous OPeNDAP servers across the Internet.

2.3 What makes a good scientific data server?

Anagram's design is influenced by several basic requirements for scientific data servers: adaptability to

* Joseph Wielgosz
COLA, 4041 Powder Mill Road, Suite 302
Calverton, MD 20705
joew@cola.iges.org

diverse network data protocols and back-end data storage systems, ease of deployment, high throughput, and stability.

In order to be useful beyond the short term, data servers will need to adapt to new protocols. While the World Wide Web has certainly changed a great deal since its creation, the protocols for distributed data systems are in a state of even more rapid evolution. Work is continuing on existing protocols such as OPeNDAP, THREDDS (Thematic Real-time Environmental Data Distributed Services), GridFTP, and others, to address newly identified needs. New geoscience oriented protocols, based on emerging standards such as the Web Services stack (SOAP, WSDL, UDDI, and so on), and the OpenGIS specifications, will likely play a large role in efforts to move towards the much-talked-about Grid computing paradigm.

An extremely wide range of data formats and analysis tools are used in geoscience data processing, with new ones appearing continuously. Each has its own desirable features, areas of specialization, and established user community. Because of this, efforts at broad standardization have not thus far been very successful. Therefore, to achieve interoperability, data servers will need to provide remote access to a wide range of existing formats and tools, via shared protocols.

Ease of deployment is also an important goal. Experience of the IT group at COLA has been that for a technology to be successfully adopted in the geoscience community, it is extremely helpful if it can be done incrementally. If a new technology requires too large an investment in learning and deployment effort, it may fall by the wayside regardless of its technical merits. Thus data server software is likely to see far more operational usage if it can be conveniently deployed by non-expert users on a variety of systems, and does not require complex configuration or maintenance.

Lastly, performance and reliability are of particular importance for machine-to-machine interfaces. While human users are limited by the speed with which they can read, click links, and so on, scripted or otherwise automated clients can easily saturate the resources of data servers. A data server therefore needs to be designed to operate efficiently under conditions of heavy and continuous use.

Human users are also able to respond adaptively to small errors or inconsistencies. For example, if a web page doesn't load the first time, we might try loading it again later, or look for the information elsewhere. By their nature, automated systems are not as versatile. Therefore, it is essential that each component of the system be as reliable and predictable as possible, both in normal operation and in its failure modes, in order for the system as a whole to function somewhat smoothly.

While the design and implementation of Anagram is far from fully addressing all of these four basic goals of flexibility, deployability, reliability, and performance, it does represent an attempt to maximize and balance them in an operational system.

3. THE ANAGRAM FRAMEWORK

3.1 Open-source technologies: Java, XML, OPeNDAP

In order to be widely useable and adaptable, Anagram is based entirely on free, open-source, platform-independent technologies.

The framework is written in Java. This provides some of the advantages of higher-level scripting languages, such as Perl, Python, PCP, etc – automatic memory management, built in multi-threading support, object-oriented environment with a rich set of standard class libraries, and minimal portability problems. At the same time, it retains a good set of low-level byte-manipulation routines as are found in C/C++, has a highly optimizing compiler, and performs well under heavy loads.

All configuration files are in XML, which eliminates the need for customized parsing routines. The Apache Jakarta Tomcat server handles the HTTP protocol details, connecting to the Anagram server via the Java Servlet interface.

The current implementation is focused on supporting the OPeNDAP data protocol, for which support is available in both C++ and Java. The open-source Java-DODS library, developed by the OPeNDAP group, is used to manage subsetting operations internally, as well as for decoding requests and encoding responses.

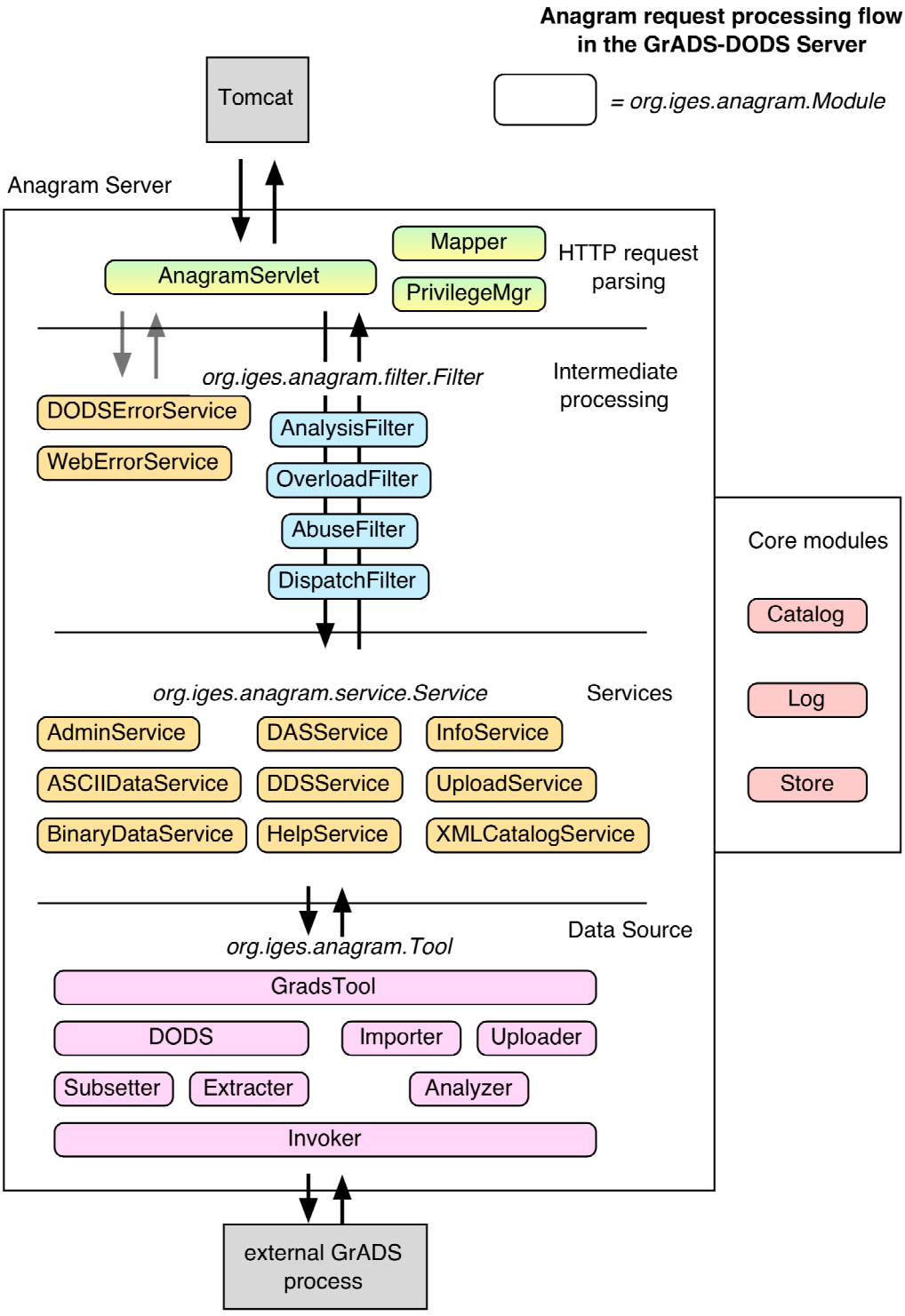
And of course, Anagram itself is available under a license that allows free reuse and distribution.

3.2 Adaptable Modules

An Anagram server consists of a hierarchical collection of Modules, each of which encapsulates a particular subsystem and may own other Modules representing subsystems of the subsystem. They are tied together partially at compile-time through factory routines, and partially at run-time through XML directives, to create a complete functioning implementation of the framework.

This approach mirrors that of the more generalized open-source Apache Avalon and Axis server frameworks (Graham, et al), and is based on well-established object-oriented design patterns (Gamma, et al). A modular design helps to reduce coupling between unrelated code areas, and facilitates tailoring the system to particular needs, and extending its capabilities into new areas.

Each module has a name, and is configured by passing it an XML fragment contained in a tag matching its name. Thus the structure of the XML configuration file for the server is simply a mirror of the internal module hierarchy.



Each Module also has a specialized interface for performing its particular task. Modules exchange small, lightweight objects in order to accomplish their tasks.

The server is started and stopped using a set of portable shell scripts, which manage the Java process and respawn it if necessary.

3.3 Layered request processing: Filters, Services, and Tools

In order for the framework to easily accommodate new protocols, and diverse analysis and data access tools, the system is divided into two major layers: a protocol layer,

and a data-format layer. In addition, a collection of core modules acts as a broker between the two, and provides access to system resources (see Figure 1).

The core modules include a Catalog, which maintains the list of data holdings in the form of a hierarchical collection of generic DataHandle objects; a Store, which provides access to a temporary working area on disk; and a Log, which handles multi-level configurable logging.

The protocol layer consists of a Mapper, a PrivilegeMgr (privilege manager), and a collection of Filters and Services. Each Request is encapsulated as a ClientRequest object when it is received through the Servlet interface. The Mapper module then assigns the incoming request to the appropriate service.

The PrivilegeMgr also assigns a privilege level according to IP address. Each privilege level maps to an XML fragment specified in the configuration file. Each Module involved in request processing will check the settings contained in the XML fragment, to determine whether a given operation should be permitted. This enables the easy the addition of new security checks at arbitrary points in the request processing chain.

After a privilege is assigned, the request is passed through the Filters, which perform intermediate processing. Anagram includes filters to turn away requests when under heavy load; block individual clients who exceed usage limits; and extract and process analysis tasks embedded in data request URLs.

Finally, the request is dispatched to its assigned Service, which generates a response. The Service has direct access to an open network socket, so that responses can be streamed back to the client as they are generated. This plays a key role in the performance and scalability of the server. The system memory needed by an open connection remains small and constant, regardless of the size of the response. Thus the server can easily handle many dozens of simultaneous connections, possibly streaming many megabytes of data to each one, without running out of system resources. An additional benefit is that the client begins receiving data for a large request immediately, without having to wait for the entire request to be marshaled on the server.

Error handling is done using the Java exception mechanism. If an exception is thrown at any time during processing, both the request and the exception are passed to an ErrorService, which sends an error response in a form appropriate to the protocol.

Anagram currently includes Service modules for the complete DODS/OPeNDAP v2.0 protocol, as well as HTML and XML-based interfaces for browsing the server catalog, basic URL-based remote administration, and data upload using the HTTP POST method.

The process of adding new Filters and Services to the system will be described in the next section.

When a client makes a request to access the server's data holdings, the assigned Service makes the necessary calls to the Tool module. The Tool interface provides a wrapper around the entire data-format-specific portion of an Anagram server. Developing a data server to access a particular data format or storage type is simply a matter of writing a Java class or set of classes that implement this interface.

One responsibility of the data-format layer is to provide the Catalog with DataHandle objects, representing the data holdings of the system. It is up to the data-format layer how these are generated and what they represent – they may represent local resources such as files or database tables, specified by configuration file directives; or they may be the result of processing client upload or analysis requests. Each DataHandle contains generic information used to generate directory listings and manage storage, as well as a separate, freeform object which can contain any fields the data-format layer will use in later processing.

The data-format layer also handles metadata and subsetting requests, either by returning JavaDODS objects for a given DataHandle, or by streaming formatted binary, ASCII or HTML directly to an open network socket for efficiency.

How these tasks are accomplished is entirely up to the data-format layer. This provides adaptability to the many different data storage formats and systems in use, and also allows for implementation-specific performance optimizations for common requests, particularly those that are data-intensive.

4. USAGE IN THE REAL WORLD

4.1 An operational Anagram server: GDS 1.2

The GrADS Data Server 1.2 consists of the standard Anagram distribution combined with an implementation of the data-format layer based on the GrADS (Gridded Analysis and Display System) analysis package (Doty and Kinter, 1995).

COLA has been running the Anagram-based GDS 1.2 operationally since spring of 2002 at multiple sites. Our most heavily used servers are currently handling over 600,000 data requests per month, and the server has been successfully deployed at a number of other sites including NOAA centers such as CDC, GFDL, NCDC, and NCEP; NASA/GSFC; and NCAR/UCAR. Current usage statistics are available at http://www.iges.org/stats/gds.

GDS 1.2 is also in use in the Land Information System (LIS), a high-resolution distributed land-modeling experiment. The server is being used to distribute forcing data inputs to a cluster of over 200 Linux-based compute nodes in real time, as well as providing a primary means of access to the final LIS model outputs.

The GrADS-based implementation of the data-format layer generates DataHandles either from GrADS descriptor files or from supported self-describing file formats such as NetCDF, HDF 4, and WMO GRIB and BUFR.

The generic field of the DataHandles is used to keep track of whether datasets contain gridded or station data, which of the four differently-linked GrADS binaries should be used to open each dataset, and whether they are self-describing or have a descriptor file. A number of metadata-related directives are also stored, such as per-dataset filters on which attributes should be passed to the client.

GrADS is newly invoked as a separate system process for each data request. A collection of GrADS scripts provides an interface to the Java modules for extracting metadata, generating subsets, and generating new datasets based on analysis requests. The Java modules manage these external processes with timeouts to prevent excessive use of resources.

This technique for connecting the Java modules of the server with an existing body of ANSI C analysis and I/O code has proven to have a number of benefits. The server is able to handle many simultaneous requests without any need to modify the C code for multi-threaded execution. The server is also protected from any potential instability or hangs in the C code, as it runs in a separate process that can be timed out if it doesn't return normally. The system has proved to port simply and smoothly to a variety of systems, avoiding the potential complications of using dynamic shared libraries. And of course it is not limited to interfacing with programs written in C, but is equally suited to FORTRAN programs, shell scripts or any other existing means of accessing data storage.

For example, an Anagram implementation based on the Ferret package has been developed at PMEL, which will be used to complement the capabilities of the Live Access Server (Rogers, Hankin and Manke 2004).

4.2 Three simple steps to using Anagram for your project

This framework was created, and this paper written, in the belief that there are likely to be many new ongoing data server projects which could benefit from having a well-developed operational framework to use as a starting point.

Potential data-format implementations for Anagram include servers that access relational database systems, invoke Java libraries, external packages, or custom FORTRAN codes to handle specialized data formats, or even retrieve data from a non-disk storage media such as mass stores.

Examples of new protocol support could include generation of THREDDS catalogs, DODS/OPeNDAP v4.0 XML messages, a SOAP/Web services interface, or generation of custom data products such as images or data files. The framework is currently oriented towards HTTP as a transport, but could potentially be adapted towards other transport mechanisms such as GridFTP as well.

Adding a new data-format implementation is extremely simple:

1. The first step is to implement the seven methods in the Tool interface - data handle creation, metadata, subsetting, and optionally analysis or uploading.

2. The second step is to compile the new Tool subsystem in to a Java Archive (.jar) file and include it in the compilation script.

3. The third and final step is to edit the XML configuration file for the Anagram web application, which contains text resources such as the server name and home page for user help, as well as the class name to load for the Tool implementation. The new server implementation is then ready to run.

Adding new Services and Filters is almost as simple. The steps are almost the same as for adding a new data-format – first write a new implementation of the Service or Filter interface, then add it to the server configuration, and finally edit and rebuild the web application. The only difference is that unlike the Tool module, which is loaded according to the class specified in the web.xml file, the Services and Filters are currently loaded at compile time. Therefore, it is necessary to subclass the Mapper, AnagramServlet, and Server classes, and replace a few methods:

Mapper.createServices()
AnagramServlet.createFilters()
AnagramServlet.init()
Server.createModules()

with code that that loads the new Services and Filters.

Replacing core modules such as the PrivilegeMgr, Catalog, or Store is also possible. Because these classes are referenced directly by other Modules rather than via generic interfaces, the new implementation must be a subclass of the default implementation. Otherwise, however, the process is no different.

5. CONCLUSION – A FIRST STEP

Anagram is an experiment in the development of distributed data infrastructure in the geoscience community, and there is much room for expanding and improving it. Due to limitations of time, and the learning process of creating the design, many areas exist which its flexibility could clearly be improved without sacrificing performance.

Some examples are the current limitation of the system to a single Tool; the awkward means of adding new Services (which ideally would be grouped together as Protocols) and replacing core modules – both of which

ideally would be configured via XML; and the HTTP-specific orientation of the core request processing classes.

The author is moving on to other projects, and thus further development at COLA is likely to be limited to operational maintenance for the immediate future. However, it is hoped that if Anagram proves useful in its current state, members of the community will be interested in expanding its capabilities, and/or addressing whatever deficiencies they encounter.

It is hoped that this experiment will inspire further exploration of the potential for the use of adaptable frameworks, based on portable, open-source technologies, in integrating the community's many diverse data archives and analysis tools into a common distributed data infrastructure.

6. RELEVANT WEBSITES

Anagram: http://www.iges.org/anagram
COLA/IGES: http://www.iges.org
GrADS: http://ww.iges.org/grads
GDS: http://www.iges.org/grads/gds
LIS: http://lis.gsfc.nasa.gov/
COLA Public GDS: http://cola8.iges.org:9191/

7. REFERENCES

Davis, E. R., and J. Gallagher, 1999: Using DODS to Access and Deliver Remote Data, *Fifteenth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology,* Dallas, TX.

Doty, B., and J. L. Kinter III, 1995: Geophysical Data Analysis and Visualization Using GrADS. *Visualization Techniques in Space and Atmospheric Sciences*, eds. E. P. Szuszczewicz and J. H. Bredekamp. (NASA, Washington, D.C.), 209-219.

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

Graham, S., S. Simeonov, T. Boubev, D. Davis, G. Daniels, Y. Nakamura, and R. Neyama, 2001: Building Web Services With Java: Making Sense of XML, SOAP, WSDL and UDDI, SAMS Publishing Indianapolis, IN

Rogers, R., S. Hankin, and A. Manke, 2004: The Ferret-DODS Server, *20th International Conference on Interactive Information and Processing Systems (IIPS) for Meteorology, Oceanography, and Hydrology*, Seattle, WA.

Wielgosz, J., B. Doty and J. Adams, 2002: The GrADS DODS Server: An Open-Source Tool for Distributed Data Access and Analysis, *Seventeenth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology,* Long Beach, CA.

Wielgosz, J., B. Doty, J. Gallagher, and D. Holloway, 2001: GrADS and DODS*, Seventeenth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology,* Albuquerque, NM.