028948-1-F

NASA EOSDIS Information Management System V0/V1 Evaluation: Final Report

Anthony W. England
Nigel Hinds

DECEMBER 1996

28948-1-F = RL-2402

NASA EOSDIS Information Management System V0/V1 Evaluation: Final Report

Nigel Hinds and Anthony W. England

Radiation Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan
Report RL-945

December, 1996

Abstract

This report summarizes our activities as members of the NASA EOSDIS IMS evaluation team in FY96. Based on our experiences we have identified a number of issues and provided recommendations. We also give a preliminary progress report on our information discovery work.

Contents

1 Introduction

An Earth Observing System Data Information System (EOSDIS) priority is to provide a collaborative environment in which science investigators can share data, models, and modeling tasks. Success of EOSDIS requires that its Information Management System (IMS) overlay present the science user a uniform and friendly interface which is relatively independent of location and nature of the data. As a principal investigator on NASA, USGS, and NSF non-EOS grants concerned with global climate change, and as a member of the Land Processes Distributed Active Archive Center (DAAC) Science Advisory Panel for the EROS Data Center (EDC), SSM/I Products Working Team (SPWT) of the National Snow and Ice Data Center (NSIDC), and the Science Advisory Group to the Polar DAAC, we would very much like to see EOSDIS succeed. Furthermore, we have been in an ideal position to assist in EOSDIS evaluation.

Over the past four years we have been funded to assist the EOS developers design and build the EOS Information Management System. Our approach to this task was partially defined by our role as "tirekickers". In that capacity we have evaluated numerous V0 and V1 IMS releases. Written feedback was provided in the form of reports as well as responses to developer surveys. In addition, we continually incorporate feedback from local users of the system. We have also evaluated many EOSDIS design specification documents and participated in technical meetings. Throughout the duration of our funding, the unique combined computer and Earth science backgrounds of the authors have allowed us to provide substantive information systems design feedback in addition to our user interface recommendations as tirekickers.

2 Issues and Recommendations

During the last year we have participated in the major design reviews (PDR and CDR) as well as a number of workshops, smaller review meeting and telecons. The Release-B CDR panel discussion format was very productive. The format gave the attendees time to discuss the system

in enough detail to provide substantive feedback. As the project continues, this format will become more important. We would encourage Hughes and NASA to continue and expand this practice.

The remainder of this document summarizes our observations and recommendations.

2.1 Web-based Interface

We strongly encourage the move toward a web based interface to NASA metadata with the development of JEST. Our experience and those of other V0/V1 tirekickers has been that non-Web versions of V0 & V1 require considerable system support to install and maintain. For example, a DCE client environment will be necessary to run the V1 client. These requirements are likely to be prohibitive for a large group of potential users outside the core group of EOS investigators. We do not consider the Guest login as a viable alternative. The current strategy of Guest login allows an unregistered user to connect to a DAAC (or other site with an ECS client), run the client at the DAAC and display graphic screens on the user's local workstation. We have argued (see Appendix A) that this approach has a number of performance issues and could become a bottleneck. The DAAC computer will run a separate process for each user that logs into the system. This means user interactions with the screen, such as button clicks and mouse moves, will generate network traffic. Any network delays will interrupt system response to the user. For example, network delays can interrupt line drawing as a user mouses a box to select a geographic area.

Clearly an easily maintainable local V1 client would be ideal, but barring that, for the reasons stated above, a Web/Java client will be an important point of entry into ECS. Web/Java is a generic client that runs at the users workstation, avoiding much of the network traffic associated with V0 or V1 running at a remote DAAC. We admit that there are a number of problems with Java and in general with Web based user interfaces. One of the more important issues is that Java only allows an applet (a program down-loaded from a Web server) to communicate with the host from which it was down-loaded. It is conceivable that this restriction could be relaxed. And we

would encourage Hughes and NASA to pursue this approach and if necessary to aid and guide the Internet community as it develops these technologies.

2.2 Application Programmer Interface

An Application Programmer Interface (API) is a set of low-level data access routines that are common across many applications. APIs are very useful in environments where many different applications are accessing similar data. ECS is an example of such an environment, where many ECS subsystems will need access to metadata and the data dictionary. Further, based on our experiences with the V0 and V1, there will likely be many flavors of the user interface. APIs will make it easier to build and evolve these applications by providing a stable infrastructure on which to build. Being one of the early advocates for such a library we are pleased to see it included in Release B. However, after a number of conversations with Hughes developers, it is clear that Hughes is not using the library to build its own client. Since the Hughes client will not directly depend on this library there is a danger that the library will not provide a complete set of functions necessary to build effective clients. This also begs the question; if Hughes is not using the API library (described in 819-RD-001-001) to build its user interface, then what is Hughes using?

2.3 Client Statistics Collection

As members of the Client Statistics collection Tiger team we made the following observations and recommendations:

- Some of the statistics can and will be captured at various EOS servers, however, if not reassembled to provide a user perspective we may loose potentially useful information.

- Once the data has been captured at the client, the existing mechanisms (e.g. email) can be used to transfer the data to the MSS. The user feedback tracking requires the client to send messages to the MSS. This mechanism should be generalized as much as possible to allow arbitrary data to be sent to the MSS.

- We realize there may be privacy issues with collecting user statistics. We recommend developing mechanisms and policies to ensure the general public can not obtain user statistics.

- Some Statistics we recommend collecting at the machine hosting the ECS client are:

  - Configuration data
  - Client failures: when and how the software crashes
  - User Context:

    When a user submits a comment while performing an operation in some cases it will be helpful (to the technician reviewing the comment) to know what the user was doing that led to the comment. This type of context data can be captured at the client and transmitted with the user comments.
  - Query response time
  - Number and frequency of searches
  - Fields used in search
  - Local cache size
  - Local cache hit rate

2.4 Client Subscriptions

As members of the Subscription Tiger team, formed at the end of September, we are making the following initial recommendations:

- The subscription system should define an event model that includes a Event Type class and Event Instance class system as described below:

  Event Type Class System:

  The event types are assigned from a hierarchical event class structure. This allows subscribers to higher level events to receive general notification messages while those processes or

users interested in more detail can subscribe to lower level events.

Event Instance Class System:
This would allow users/processes to receive events grouped by application. This will require an Event Type Class and an Event Instance Class. So, an event would have a type attribute, such as Invalid-Input and a group parent attribute, such as MODIS-DATA-PRODUCT1.

- Size and location of subscriptions can have a major impact on system performance. The project should consider architectures that include centralized as well as distributed subscription management, including mechanisms that allow subscription consumers (or their agents) to assume considerable management responsibility.

2.5 Catalog Interoperability

Many of the catalog interoperability issues overlap our work on distributed information discovery. As a result we have provided feedback to the catalog interoperability protocol work. We summarize that feedback below.

We recommend using the Web as the substrate for any distributed information system. There are the obvious installed user-base advantages. Further, HTTPD's CGI provides a flexible mechanism on which to build fairly complex database interfaces.

The serious drawback is that current Web indexing uses a "pull" approach where custom robots search for data. In some cases it will be more efficient to notify brokers of new or modified data ("push"). This can be accomplished with companion services that run along side HTTPD, using JAVA for example.

Information Discovery Status:

Our information discovery work suggests that it may be possible to use hierarchical subject indices, such as the GCMD, to reduce search times for distributed data. Over the last year we have developed the Domain Metadata Service (DMS) design and prototype. The DMS architecture uses a version of the IMS parameter list to construct centroids. Centroids are synthetic data averages; they are, in effect, higher levels of metadata. Because centroids are smaller than the original data and metadata, more of them can be stored in the same amount of space. Together with pointers to archive sites, the centroids make-up referrals. Referrals form a distributed data structure used to direct the DMS distributed search engine. The prototype is currently being tested, the results of which will be published next year.

3 Conclusions

This report has summarized our activities as members of the NASA EOSDIS IMS evaluation team. Based on our experiences we have identified a number of issues and provided recommendations. Finally, we gave a preliminary progress report on our information discovery work.

A MB-IMS

An Experimental Mosaic Interface to Scientific Information Systems

Nigel Hinds and A. W. England
Radiation Laboratory
The University of Michigan
Ann Arbor, Michigan 48109-2122, USA
nigel@eecs.umich.edu, england@eecs.umich.edu
Phone: (313) 763-5243
Fax: (313) 747-2106

ABSTRACT

The NASA Earth Observing System (EOS) information system, EOSDIS[1], is a key component for disseminating EOS instrument data. Within EOSDIS the Information Management System (IMS) provides access to products maintained at data archives distributed throughout the country. At the heart of the IMS graphical user interface is the widely accepted X11 display protocol. An important feature of X11 that has facilitated IMS prototyping is that it allows users to separate the host computer on which an application program runs from the host where the output is displayed.

Running remote X11 applications in this manner is appealing to organizations that for one reason or another do not want to distribute and support software packages or release the source code for their applications. The drawback is the X11 network message traffic generated when an application is run remotely.

Our work investigates the feasibility of eliminating X11 network traffic by using HTML+ and Mosaic to configure interfaces to information systems such as the NASA IMS. This approach offers advantages to both the information supplier and consumer. To the information consumer the system provides a single X11 software package which can communicate with any World Wide Web (WWW) information supplier. For the information supplier the approach will eliminate the need for X11 programming.

INTRODUCTION & MOTIVATIONS

There are many network data systems which provide access to a wealth of information [5]. Most of these systems offer only simple character-based user interfaces. The X11 window system has created a new generation of sophisticated graphical user interfaces.

[1]Earth Observing System Distributed Information System

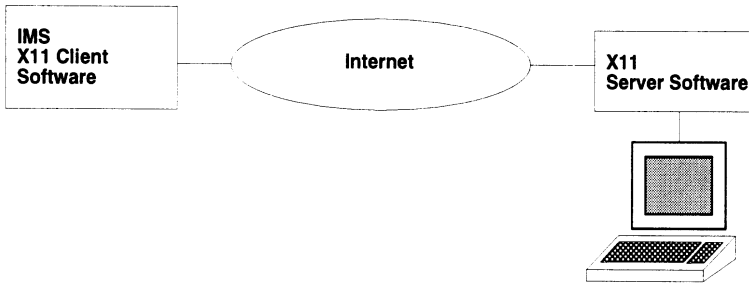

Figure 1: X11 Remote Server Model

Assuming each user has a graphics workstation running X11, then application software can run on the user's workstation, or on a remote host. X11 accomplishes this by separating the system that displays data on a screen from the application that makes the display requests. In the X11 model the screen on any workstation is called the X11 server which accepts display commands on a designated communication port, and the computer running the application is considered the X11 client.[2] The X11 application is also commonly referred to as the client.

Although running applications on the host with the X11 server avoids network delays, many organizations do not have the mechanism to distribute and support software packages. Furthermore, administrative, security, and licensing issues may make it impossible for these organizations to release their source code. Even if source code were released, those users having the expertise to configure and compile it for their computing environment could potentially have to compile a new X11 client for each information server they wanted to

[2]We should note that some consider X11's client/server model to conflict with the traditional client/server definition. Typically, in distributed data systems, a *client* process on the user's computer sends a request to a data *server*. In X11 the user's computer receives display commands and is called the display *server* while the computer sending the display commands is called the *client*.

contact. For these organizations, running applications remotely is an appealing alternative. Figure 1 uses the NASA EOSDIS-IMS graphical user interface to illustrate the X11 remote execution model. In figure 1, the IMS user and the IMS software are located at different network sites. The user's computer is equipped with a graphics computer running the X11 server which allows it to function as a display for any X11 client. On the other side, the IMS software runs as an X11 client program. To operate the system, the IMS user logs into the IMS client computer and starts the IMS X11 client program, instructing the software to display the screen images on the user's local X11 server. Unfortunately in this model almost all user input including mouse movements at the local X11 server is sent back through the network to the X11 client computer. So, even minor network delays can profoundly affect the user interface and frustrate users.

To get rough figures of the network traffic generated by running the IMS remotely, we counted the bytes transferred from the X11 client and server. An IMS session was initiated and Sun *Etherfind* was run to monitor traffic between the application run at Goddard Space Flight Center (GSFC) and the X11 server at the University of Michigan. Once the application was running, the experiment consisted of repeatedly displaying and closing the IMS search window (Figure 2). The bytes received by the local X11 server ranged between 20,120 and 20,444. The number of packets ranged between 61 and 67. Although 20K bytes does not appear significant, network performance can have a major impact on the time it takes to display a screen.

The new generation of information systems such as the Wide Area Information Server (WAIS) [7], Gopher [6], and World Wide Web (WWW) [2] address the issue of multiple X11 clients as well as network traffic. Each of these systems define a traditional client-server architecture for building distributed data servers. For example, WAIS uses the Z.39.50 [7] protocol to describe how client programs and WAIS servers communicate. Similarly, WWW uses HTTP and HTML [2] to describe client-server communication. By defining a standard client-server protocol, these systems make it possible to provide the user with one interface application that is capable of communicating with any data server system using the protocol standard. Since the user now only needs one application, administration and support are simplified and building an X11 application which runs locally is much more feasible. Unfortunately, because each of these systems use a different client-server protocol, the user is once again faced with having to use three different applications to access data stored in WAIS, Gopher, and WWW. However, supporting three powerful applications is an improvement over the scenario described in the Introduction.

At the University of Michigan we are exploring the feasibility of implementing the IMS forms-based user interface in WWW. We chose to start with the WWW architecture and its X11 interface application Mosaic because their Hypertext Markup Language (HTML) was very easy to use and had the form building features we needed.

DESIGN

Hypertext Markup Language (HTML) [3] is based on the Standard Generalized Markup Language (SGML) [1]. Both languages annotate plaintext documents with tags to impose logical structure. Examples of tags are chapter and section headings, item lists, and character highlight.

In the WWW system, authors use HTML to construct documents stored at their WWW server. HTML documents along with text and other multimedia may contain hypertext links (hyperlinks) to other HTML documents at any WWW server. Users access WWW documents with the Mosaic X11 application. Mosaic is the X11 client to the user's X11 server, as well as a client to the WWW data server. The single Mosaic X11 application reads HTML files retrieved from any WWW server and displays them for the user. This Mosaic X11 application runs locally and only generates network traffic when it sends queries to the WWW servers.

Mosaic with the fill-in forms additions of HTML+ [4] met many of our information system requirements. HTML+ added input fields, radio button fields, and Action buttons which send queries to WWW servers. Figure 3 shows a portion of the prototype IMS search screen document we have constructed using HTML+. A WWW server has been configured to accept queries generated by the IMS document and return results from a test database.

STATUS & RESULTS

HTML+ allowed us to include many of the basic forms features we needed in the prototype. However, HTML being only a logical markup language, it does not provide layout directives. Aesthetically the result left something to be desired. Also, we encountered a number of features which could not be rendered in our HTML+ document. Many of the problems are due to the document viewing model used by HTML/Mosaic. It does not permit sophisticated interaction with the user. We outline some of our difficulties below.

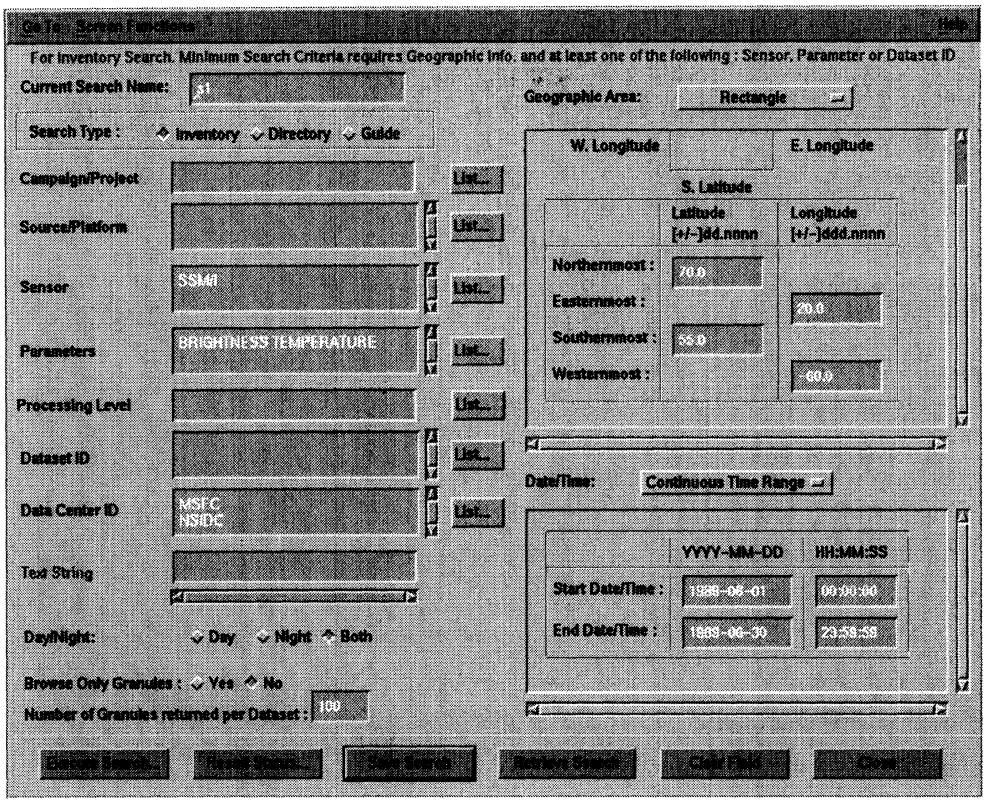

Figure 2: IMS Search Screen. This image consumes almost all the area on a 17 inch screen.

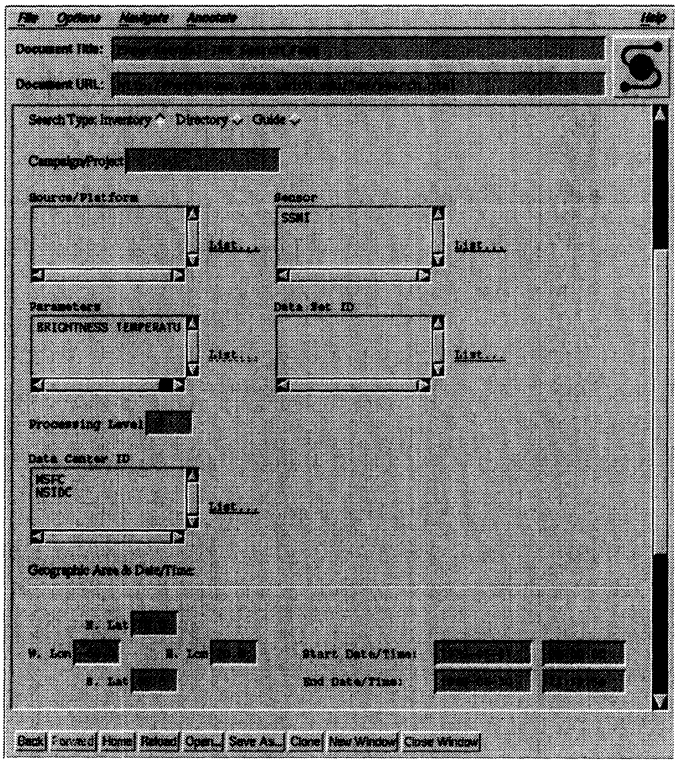

Figure 3: Mosaic IMS Prototype

- The original IMS (Figure 2) uses popup menu choices to reconfigure the screen. For example, the fields displayed in the *Geographic Area* depend on the item selected from the menu. In Figure 2 the menu reflects our current choice, *Rectangle*, which displays the lat/lon input fields you see in the figure. We could have also chosen *Point Radius* which would have displayed a different set of input fields. HTML+ does not provide tags to describe such interaction.

- The original IMS allows users to fill fields from a list of valid input choices. Using a popup scroll list users may select items which are then automatically entered in the field.

  For example, in the current NASA IMS prototype there are seven valid data centers which the user may view and select in a popup list by clicking on the *List...* button next to the Data Center field in Figure 2. Although hyperlinks (displayed as *List...* in Figure 3) allow us to display another document containing valid entries, once a user makes a selection there is no way to reflect that selection back on the search document.

  Also, it is not possible to implement the notion of *dependent valids*. With dependent valids, the list of valid input choices is further restricted by the

input the user has already entered. For example, the *parameter* field valids list would not display *WIND SPEED* if *SSM/I* had already been entered in the sensor field.

- HTML+ has only *Submit* and *Reset* action buttons. Submit sends a list of all the fields and their values to the server where the document is stored. The server evaluates the query and returns the results in a new document. Reset clears the fields in the document. Arbitrary action buttons could allow us to implement dependent valids. For example, with a *List...* action button, the current value of all fields could be sent to the WWW server to restrict the valids lists displayed for the field associated with the List button.

- The interaction with WWW and Mosaic is a synchronous client-server model. Mosaic sends HTML requests to the WWW server and waits for a response. The model does not allow the WWW server to notify the Mosaic system or otherwise initiate communication. This would be very useful in the event of a time consuming search. Instead of waiting for a response from the WWW server Mosaic could allow the user to read other documents until the search results were ready.

- With generic hypertext documents it might be hard to anticipate usage, but with information systems the navigation paths can be predictable. It might be faster to download a number of pages at once. That way, hyperlinked documents could access them without communicating with the WWW server.

CONCLUSIONS

There are clear benefits of co-locating X11 applications with their X11 display servers. However, this is not possible when there are large numbers of information providers requiring separate user interfaces. Our preliminary results demonstrate the feasibility of providing a generic information system interface with HTML+ and Mosaic. Our current plans are to complete the IMS search screen page and compare the Mosaic network traffic to the X11 traffic discussed in the introduction.

Even with its features, HTML+ and Mosaic will not allow us to faithfully recreate all aspects of the IMS screen. As we go on to implement more IMS screens we anticipate proposing modifications to HTML+ and Mosaic to overcome some of the issues mentioned in the last section.

ACKNOWLEDGMENTS

This work has been supported by a grant from the EOSDIS project through the Goddard Space Flight Center.

REFERENCES

[1] ISO 8879:1986. *Informaion Processing Text and Office Systems Standard Generalized Markup Language (SGML).* International Standards Organization, 1986.

[2] T. Berners-Lee, R. Cailliau, J. Groff, and B. Pollermann. World-Wide Web: The information universe. *Networking: Research, Applications and Policy*, 2(1):52–58, Spring 1992.

[3] Tim Berners-Lee and Daniel Connolly. Hypertext markup language. Technical report, CERN and Atrium Technology Inc., July 1993.

[4] IETF. HTML+ (hypertext markup format). Technical report, IETF, November 1993.

[5] J. Martin. There's Gold in Them Thar Networks! or Searching for Treasure in all the Wrong Places. RFC-1290, Ohio State University, December 1991.

[6] M. McCahill. The internet Gopher: A distributed server information system. *ConneXions - The Interoperability Report*, 6(7):10–14, July 1992.

[7] TMC. WAIS interface protocol, prototype functional specification. Technical report, Thinking Machines Coporation, 1988.