HIRDLS

TC-UCB-060

HIGH RESOLUTION DYNAMICS LIMB SOUNDER

Originator: Ken Stone

Date: **Prior to 2004**

Subject/Title: **The HIRDLS Science Investigator-led Processing System (SIPS): Supporting operational processing and more**

2009-04-30 Linda Henderson found this document without a doc. # or cover sheet. She is assuming it is the document created by Ken Stone for TC-UCB-060

Keywords:

Purpose of this Document:

Reviewed by: Date (yy-mm-dd):			

Oxford University Atmospheric, Oceanic & Planetary Physics, Clarendon Laboratory, Parks Road, OXFORD OXI 3PU, United Kingdom.

University of Colorado at Boulder Center for Limb Atmospheric Sounding, 3450 Mitchell Lane, Bldg. FL-0, Boulder, Colorado, 80301-2296, United States of America.

EOS

The HIRDLS Science Investigator-led Processing System (SIPS): Supporting operational processing and more

Kenneth Stone, John C. Gille, Tom Lauren, Greg Young
University of Colorado Center for Lower Atmospheric Studies, 3300 Mitchell Lane, Boulder CO 80301;

ABSTRACT

The High Resolution Dynamics Limb Sounder (HIRDLS) Science Investigator-led Processing System (SIPS) is a software framework designed for operational, event-driven, instrument data processing. At its foundation are multiple open-source components that are assembled into a distributed architecture. The framework allows creation and modification of a wide array of processor and product scenarios. Java is used as the primary software language though processors can be implemented with any technology supported on the target platform. The software development approach, design and implementation technology are described along with several features and benefits of the system.

Keywords: HIRDLS, Java, Science Data Processing, Open-source, NASA, DAAC.

1. INTRODUCTION

1.1 Overview

The High Resolution Dynamics Limb Sounder (HIRDLS) Science Investigator-led Processing System (SIPS) is a NASA Earth Science Data and Information System (ESDIS) funded project. Its primary goal is to process HIRDLS data in a prompt and efficient manner and subsequently deliver science products (data files) to the Goddard Earth Sciences Distributed Active Archive Center (GES DAAC) for eventual use by the scientific community. Another goal, nearly as important as the first, is to provide fast and accurate feedback to science software developers as to the status of processes and products. Together these two goals form the basis for the HIRDLS SIPS and in this paper we describe the approach, design considerations, implementation choices and several features and benefits. We also briefly describe other potential uses for such a system in the development of science data processing software.

1.2 HIRDLS

HIRDLS is an infrared limb-scanning radiometer designed to sound the upper troposphere, stratosphere, and mesosphere to determine temperature; the concentrations of O_3, H_2O, CH_4, N_2O, NO_2, HNO_3, N_2O_5, $ClONO_2$, $CFCl_2$, $CFCl_3$, and aerosols; and the locations of polar stratospheric clouds and cloud tops. The goals are to provide sounding observations with horizontal and vertical resolution superior to that previously obtained; to observe the lower stratosphere with improved sensitivity and accuracy; and to improve understanding of atmospheric processes. HIRDLS will fly on the Aura mission, part of NASA's Earth Observing System (EOS).

On the Aura satellite, HIRDLS will be pointed in the anti-velocity direction and in a nominal configuration it will scan the atmosphere at seven azimuth angles to form seven vertical profiles of radiances every 66 seconds. This equates to approximately 9000 vertical scans (sampled from ground to 120 km) per day, with complete global coverage taking around 12 hours.

HIRDLS is a joint US-UK development effort, with sponsorship by the British National Space Centre and the Natural Environment Research Council in the UK, and by NASA in the US.

1.3 Data processing

HIRDLS data will arrive at the GES DAAC in the form of telemetry packets (L0 data), which are 832 bytes each spaced 0.096 seconds apart. This results in about 750 MB of data in 24 hours. The data are formed into 2-hour granules and

delivered to the SIPS along with any associated ancillary data such as engineering housekeeping data and satellite orbit/attitude values. Figure 1 shows an overview of the SIPS external interfaces.

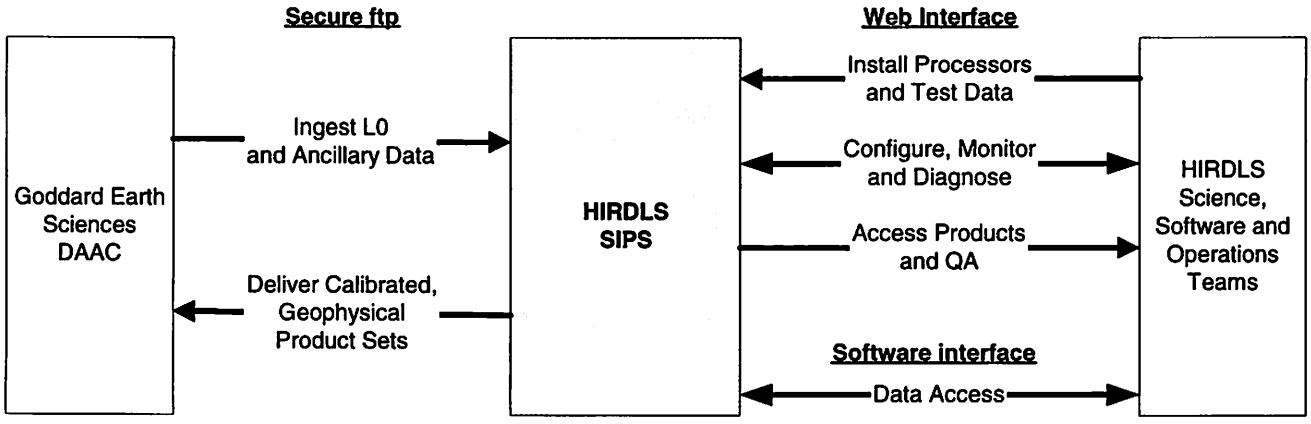

Figure 1 The HIRDLS SIPS external interfaces.

HIRDLS uses three major processing stages in moving from telemetry to geophysical parameters: L1, L2 and L3. A science data processor is associated with each product level and these processors are delivered from the HIRDLS Science Computing Facility (SCF). The L1 product contains calibrated, geo-located radiances along with pointing information. The L2 product contains geo-located geophysical parameters such as temperature, mixing ratios and aerosol extinctions. Finally the L3 product contains L2 product gridded onto a 3-d latitude, longitude and pressure grid. Along with standard products, often times processing levels will need to generate ancillary products such as diagnostics and quick look summaries. These can either be generated directly by the processor or by a subordinate processor that is installed in SIPS along with the standard processor.

In a typical processing scenario, 14 (24 hours of data plus a granule on each side of the date boundary) L0 granules will be used with ancillary data as input into the L1 processor. Some ancillary and the L1 product are used as input into L2. The output from L2 is used in the L3 processor. All of the products after L0 are nominally 24 hours in length.

After data products are generated and quality-checked, the products are delivered back to the DAAC for wider distribution to the scientific community. This involves staging the data and assembling and fine-tuning EOS metadata in order for the products to be ingested into the DAAC database.

2. REQUIREMENTS

Emphasis was placed on using industry best practices for capturing requirements and planning development, particularly as outlined by McConnell[1,2]. The requirements (and initial software development plan) were assessed at a planning review. The plan along with associated project documentation was made available on an internal web site to increase visibility. The first challenge was identifying the areas where requirements were unstable and managing those areas with the most uncertainty. The second challenge was keeping derived requirements continually in mind as development progressed.

2.1 External interfaces: GES DAAC and HIRDLS SCF

The interface requirements between SIPS and GES DAAC were well documented and continue to be relatively stable. This small degree of uncertainly provided an opportunity to implement a more traditional software life-cycle model, with less emphasis on change management.

However, the requirements for the SCF interface were expected to evolve as developers began using the SIPS, their own processors and their processors within SIPS. Science software development is frequently an iterative process. A limited understanding of the problem is used to define a software solution and results of the software are analyzed to increase the depth of understanding in order to code a new solution. This process requires the feedback cycle to be quick and accurate. At the least, the processing system cannot get in the way of the feedback. In the best case, the system facilitates feedback and understanding. This larger degree of uncertainty required a more iterative approach with the ability to quickly respond to change.

2.2 Derived

Due to the size and nature of the project, it was critical for the HIRDLS SIPS to be easy to use and simple to maintain. Although these seem like obvious qualities, they need to be kept in mind during system development. Due to the uncertainty in the SCF interface requirements, portability was also considered critical.

3. APPROACH

3.1 Awareness of requirements (in)stability

The problem of how to deal with changing requirements has received much attention in recent times, especially in the commercial software development world. Although the primary requirements for the SIPS are well-defined and stable, an understanding of the implications of those requirements on the system is evolving. This evolution in understanding has necessitated some significant changes during the course of SIPS development.

A greater force for change has come from evolving internal requirements. In addition to the SIPS' role as a production system, it must support processor developers to some extent and must support local science investigators who need access to data.

The approach to dealing with changing requirements is to use designs that are flexible, to use automated tests, and to stream-line the development process.

3.2 Emphasis on architecture

A distributed architecture and object-oriented design were chosen to provide a basis for meeting the requirements and to provide flexibility for future enhancements. The processes or components of a distributed system may be deployed on different computers on a network, allowing for example a host to be free from all but a single demanding component. Object-oriented analysis and design is a widely used development approach where the system is modeled as a collection of interacting objects. See section 4 for more information on the SIPS architecture and design.

3.3 Adopting best practices

Some software development practices were adopted that have proven helpful on other projects. Rather than adopt a particular development method (Rational, Extreme Programming etc.) as a whole, ideas have been selected which are most applicable to this particular project (considering requirements, time to launch, development team size and experience.)

3.3.1 Unit testing

Unit testing is an old but very powerful idea that has received renewed attention recently. The development of the HIRDLS SIPS relies heavily on unit testing and a Java framework called JUnit (http://www.junit.org) to insure that objects work as expected and continue to work as the code base is changed. A good suite of unit tests gives developers confidence that they can change the implementation of objects without fear of regression. This confidence is critical to being able to respond to refinements or changes to requirements.

3.3.2 Frequent integration

SIPS development relies on frequent integration of code changes into the base so that any effects on the system not caught in automated test might be noticed as soon as possible. Also, features that involve human interaction such as GUI changes may be evaluated before being seen by formal test or before being put into production.

To support this practice, a strong ethic is maintained which requires code changes to compile and to not break any existing automated tests. It is also encouraged that new features are supported by new unit or system tests.

3.3.3 To-do list driven development

Development tasks are chosen from a list of known tasks called the 'to-do list.' Tasks for developers are chosen considering the task's priority and its impact on the system. The priority of the task is determined by first considering whether or not the task is a fundamental or wide-reaching change. Such tasks are given the highest possible priority on the belief that fundamental changes should be settled as soon before launch as possible.

If there are no fundamental changes, then tasks are selected by considering if completing the task is necessary to satisfy an upcoming test by NASA (which tests the DAAC to SIPS interface) or secondly, by considering if the task is necessary to satisfy an internal requirement.

This simple approach has proven very effective for HIRDLS SIPS development because of the small team size and because external accounting requirements are not extensive.

4. ARCHITECTURE AND DESIGN

The HIRDLS SIPS uses a component based and distributed architecture as a basis for satisfying the requirements. Figure 2 shows an overview of the HIRDLS SIPS architecture.

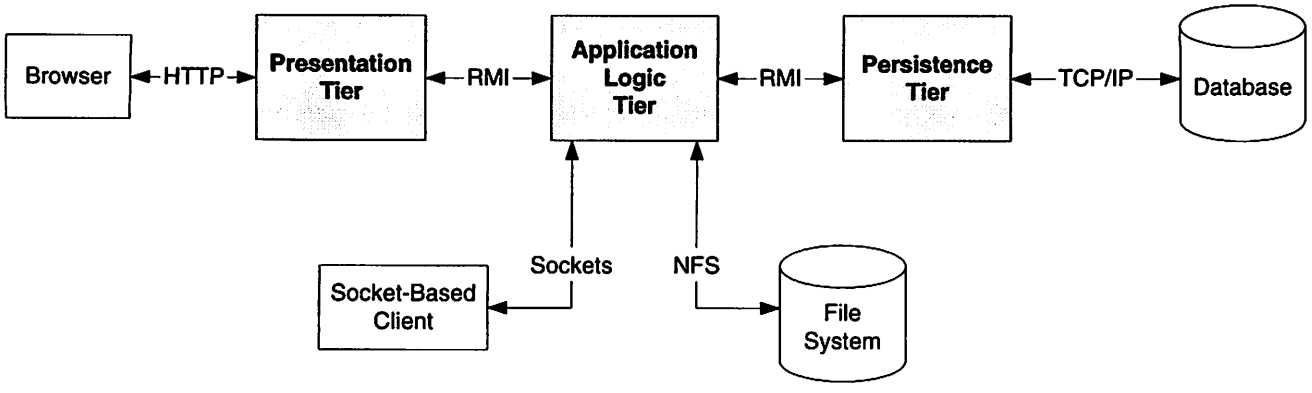

Figure 2 The HIRDLS SIPS architecture.

At the highest level, the HIRDLS SIPS has a three-tier architecture. The presentation tier is responsible for providing a Web-based view of the application. The application logic tier is responsible for the core logic of the application. The persistence tier is responsible for persisting SIPS-specific data. Each tier is capable of being installed and deployed on a different machine. The tiers communicate with each other through sockets or Remote Method Invocation (RMI). RMI will be discussed in section 5.1.2.

The application logic tier can communicate with Java and non-Java clients. By far the most expansive interface to the application logic tier is a Java-based interface relying on RMI. The presentation tier uses this interface to handle interactions with users working through a Web browser. Alternatively, applications that are not Web-based may communicate with the application logic tier through a more limited socket-based interface.

The distributed aspect of the architecture allows different parts of the application to run on different computer hosts if necessary, to either minimize the SIPS dependency on a host or to maximize the efficiency of the host for the component.

The SIPS design aims to satisfy the current requirements and to provide flexibility for extending the system to meet new and evolving requirements.

4.1 Component based

The application logic tier is component-based. Components as defined by this application are objects in an object-oriented system having some additional qualities. They are designed for re-use (are general and configurable) and use interfaces as the primary technique for object-to-object communication. Interfaces allow system designers to concentrate on the essential contract for communication and hide all other aspects of the object.

A component-based architecture allows problems to be partitioned for testability, facilitates concurrent development, and allows code to be replaced with a minimal impact on the system. Figure 3 shows an overview of the components in the application logic tier.

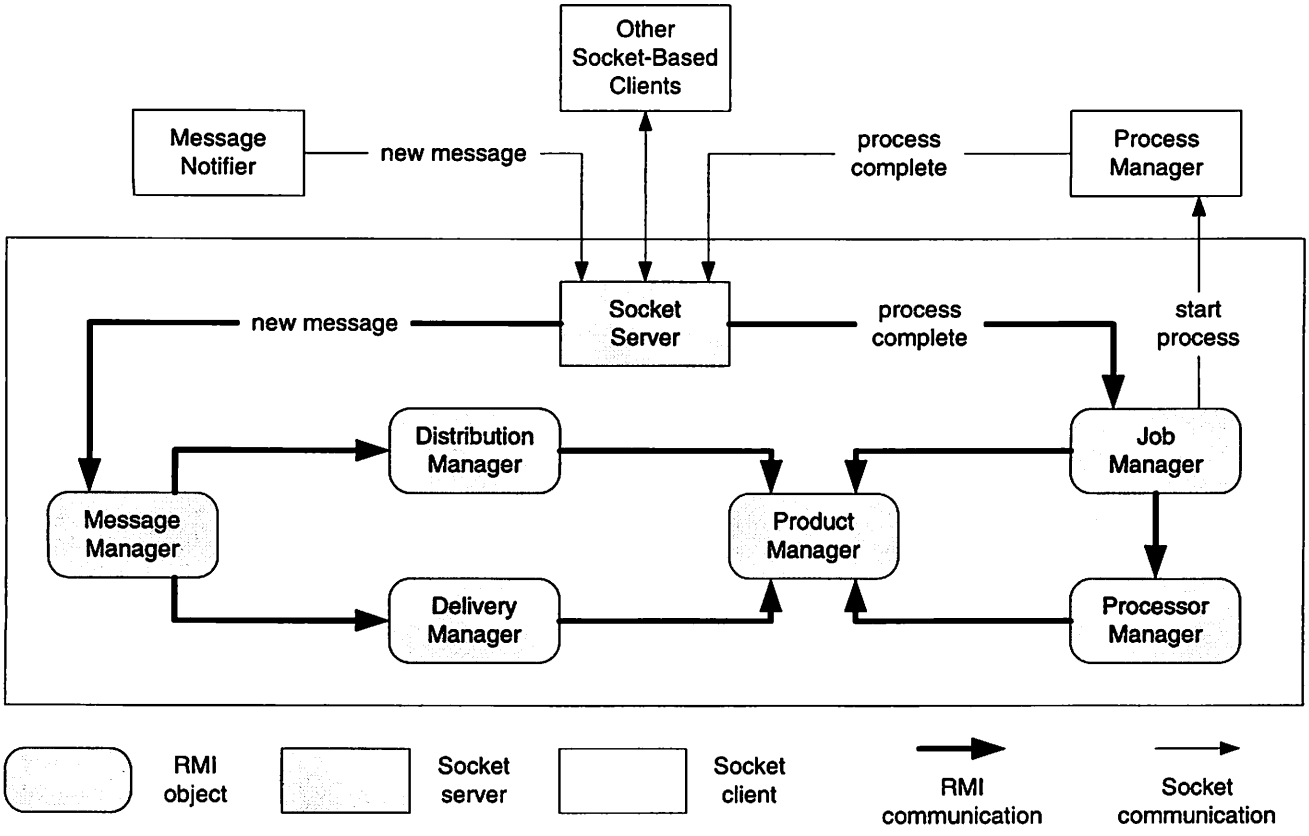

Figure 3 Components of the application logic tier.

The application logic tier is composed of several different components, each with its own set of responsibilities. The message manager is responsible for handling incoming email messages from the DAAC. The distribution manager is responsible for handling data distributions from the DAAC. The delivery manager is responsible for delivering files to the DAAC. The job manager is responsible for running multiple-processor jobs. The processor manager is responsible for installing and staging processors. The product manager is responsible for storing/retrieving files to/from near line storage.

Through RMI, the presentation tier has access to each manager component. Non-java clients can communicate with the application logic tier via a socket server component. Two socket-based clients are an integral part of the SIPS design. A message notifier client notifies SIPS when new messages arrive from the DAAC. A process manager client notifies

SIPS when processors have stopped running. The socket server communicates with these clients via XML. This allows an arbitrary number of other socket-based clients to communicate with the HIRDLS SIPS in the future.

4.2 Configurable

The SIPS is highly configurable via XML property files. This allows components to be run in different environments (hosts, operating systems etc.) by referring to different property files. Defining run-time constants as properties eliminates the need to change component code in order to run in a different environment. For example, here is a snip of XML that specifies information about the SIPS email inbox:

```
<inbox>
        <host>sips-mail-host</host>
        <port>110</port>
        <login>gdaac-login</login>
        <password>gdaac-password</password>
        <poll-frequency>600000</poll-frequency>
        <poll-startup-delay>10000</poll-startup-delay>
        <server-type>pop3</server-type>
</inbox>
```

4.3 Event driven

The SIPS is an event-driven system, with events originating from the DAAC, the SIPS GUI or an asynchronous task such as a running job. Events originating from the GUI are handled by public, remote interfaces implemented by primary components. Events originating from asynchronous, internal threads are handled by internal interfaces that may be remote or local. Events originating from the DAAC are processed by "pluggable" email handlers.

4.4 Plug-in solutions

Email events are one way (the other being ftp) to receive events from the DAAC. To allow flexibility in handling these events, the SIPS uses "pluggable" email event handlers. An email event handler must implement a particular interface and is associated with a "sender address" and a "subject." The association is made via elements in an XML property file. For example, here is a distribution notice handler:

```
<message-handler>
        <class>edu.ucar.hirdls.sips.message_mgr.DnHandler</class>
        <sender>daac-addr</sender>
        <subject-reg-exp>^ECS Notification$</subject-reg-exp>
</message-handler>
```

The class "DnHandler" implements the required interface and receives messages from "daac-addr" where the subject matches "^ECS Notification$."

4.5 Web GUI

The SIPS architecture calls for a Graphical User Interface (GUI) based on the HyperText Markup Language (HTML) and HyperText Transfer Protocol (HTTP). HTML/HTTP GUIs separate the process rendering the 'view' on the application from the process that generates the view. The HTML defining the view is generated by the SIPS web server. Users access the GUI via their favorite browser by typing a URL such as "http://sips". The principal benefit of this approach is that no software has to be installed on a user's computer.

5. CONSTRUCTION

5.1 Technology choices

The construction of the HIRDLS SIPS involves the culmination of several different software technologies. Each of the three architectural tiers within SIPS has its own responsibilities, and therefore its own technical challenges and

requirements. What follows is a discussion of the technology choices within SIPS as they relate to the persistence tier, application logic tier, and presentation tier.

5.1.1 Persistence tier

The persistence tier is responsible for storing SIPS-specific data. The technical requirements of the persistence tier include reliability, data integrity, concurrency, performance, and loose-coupling with the rest of the SIPS application. One of the most popular persistent storage technologies today is a relational database. A relational database provides many advantages over other techniques to persist data, including a powerful SQL language to perform queries, transactional support, concurrency support, security, robustness, and performance[3]. For these reasons a relational database was chosen to store SIPS data.

Although relational database technology solved the problem of where data will be stored, a clean and efficient interface between the database and the rest of the application was still required. To solve this problem, Enterprise JavaBean (EJB) technologies were introduced to the project. EJBs are customizable software components that can represent objects in a data store[4]. Together with an EJB container, EJBs provide the ability to store, update, and delete data in a transactional manner without the need to write complicated, error-prone SQL statements. With a persistence framework built on EJB technology connected to a relational database, the technical requirements of the persistent tier were met.

5.1.2 Application logic tier

The application logic tier is where the core actions of SIPS take place. The technical requirements of the application logic tier include remote access, support for multiple client types, distributable components, testability, and file-based configuration. The requirement for remote access is solved differently depending on the client type. Remote Method Invocation (RMI) is a technology that allows Java code to communicate with other Java code that is running within a different virtual machine, possibly on a different host[5]. RMI technology permits the application logic tier to consist of several distinct distributable components, with each component dedicated to handle specific responsibilities. A distributable architecture provides the ability to deploy any component on any physical server, allowing for load balancing and increased performance. Distributing components via RMI also improves the testability of the application, allowing each major component to be tested individually. Although RMI allows a remote Java client to communicate with the application logic tier, a different technology is needed to allow connection with non-Java clients. Socket technology is a well-proven mechanism for creating a virtual connection between processes. To communicate with a non-Java client, the application logic tier includes a socket server that handles XML-based requests and responses. A final technical requirement of the application logic tier is file-based configuration. SIPS was designed to be configurable through property files. Extensible Markup Language (XML) technology is a universal and portable format for describing data. Due to XML being easy to read, easy to parse, non-proprietary, platform and language neutral, and well supported[6], SIPS uses XML technology for both configuration and socket-based communication. With the help of technologies such as RMI, sockets, and XML, SIPS was able to meet the requirements of the application logic tier.

5.1.3 Presentation tier

The final tier of the SIPS architecture is the presentation tier. The technical requirements of the presentation tier include remote access from a browser, dynamically generated content, scalability, and isolation from the application logic tier. HTML and HTTP are fundamental technologies for Web-based applications. SIPS builds on these technologies to provide dynamically generated content with the use of Java Servlet and JavaServer Page (JSP) technologies. Servlets are programs that run in a Web server, handling responses/requests to/from a Web browser. Servlets provide several advantages over traditional CGI technologies, including faster response time, a smaller footprint, and the convenience of the Java programming language[7]. JSP technology enables the mixing of HTML content with dynamically generated content from servlets, separating the presentation of a Web client from the control logic[8]. Through technologies such as HTML, HTTP, Java Servlets, and JavaServer Pages, SIPS met the requirements of the presentation tier.

5.2 Open source components

To facilitate the construction of the HIRDLS SIPS, several open source components were introduced to the project. Open source software provides several advantages over alternatives such as commercially-purchased software or developing software on one's own. The advantages include collaborative development by experts in a specific domain, the ability for users to view and modify the code, a high standard of quality, fast turnaround time, long-term viability,

and free access[9]. The HIRDLS SIPS takes advantage of open source components such as a MySQL database server, JBoss application server, Xerces XML parser, Log4j logger, Tomcat web server, and Struts Web application framework.

6. APPLICATION FEATURES AND REALIZED BENEFITS

6.1 Highly configurable

The SIPS application is designed to be a fairly generic processing system. Distribution handling, file types, processors, jobs, and deliveries can all be configured for a particular instrument at run time.

The SIPS application regularly receives distributed files from a DAAC. Through a combination of Earth Science Data Type (ESDT) names and filename-matching regular expressions, the application provides the ability for an operator to configure SIPS for understanding distribution notices.

In addition to configuring SIPS for distribution handling, a user may configure SIPS to understand the various file types that are involved in a production environment. The application classifies each data file as belonging to certain type, called a file type. The application allows an operator to associate certain information with a file type, such as a short name, file suffix, description, granule length, time reader, and file storage path. This information provides SIPS with the knowledge that it needs to store the file, provide completeness information, and extract data times.

One of the most powerful configurable aspects of the HIRDLS SIPS is the installation and configuration of a processor. To install a processor, a user specifies information such as an installation version, directory, and executable. To configure a processor, a user specifies information such as a configuration name, configuration template file, arguments to the processor, and configurations for the input, output, and log files related to the processor. This information gives SIPS the ability to stage a processor for running, and helps SIPS find a correct set of file instances to use as input. The ability to install and configure new processors at run-time keeps the SIPS application loosely coupled to the individual processors that are actually run by SIPS.

The application also allows a user to configure a job. The SIPS notion of a job is one or more processors connected together by files. This gives a user the flexibility to choose the granularity of a job. A job may be one self-contained processor, or a combination of many processors, each with a smaller set of responsibilities.

Deliveries are configurable as well. The application allows an operator to define the various delivery file types that may be included in a Product Delivery Record (PDR). Delivery file types may then be associated with file types to allow deliveries to be made to the DAAC.

6.2 Production monitoring and history

The SIPS application provides a large amount of production monitoring power. Information about all distributions, file instances, processor instances, job instances, and deliveries is preserved and searchable.

The application gives a user the ability to query for and view information about any file distribution from a DAAC. One can view such details as the arrival time, granules, granule files, and any failures that may have occurred.

In addition to distribution monitoring, the SIPS allows a user to monitor files. A particular instance of a file type is called a file instance. The application gives a user the ability to query for and view information about any file instance in near line storage. One can view such details as the time that the file was produced, the file size, the arrival method (distribution, upload, or processor-generated), the data time coverage for the file, and any related file instances such as metadata.

When a configured processor is run by SIPS, a processor instance is produced. The application gives a user the ability to query for and view information about any processor run. One can view such details as the data time range, when the processor started, how long it has been running, its current state and status, who started the processor, and all of the

input, log, and output file instances associated with the processor instance. For a running processor, a user may even view platform-specific process information for that run.

Once a configured job is started, a job instance is produced. The application gives a user the ability to query for and view information about any job run. One can view such details as the data time range, when the job was started, how long it has been running for, its current state and status, who started the job, and the current processor instance.

Once a user makes a delivery request, a delivery instance is created. The application gives a user the ability to query for and view information about any delivery that was sent to the DAAC. One can view such details as the delivery creation time, its current state, the current PDR instance, and who sent the delivery.

6.3 The concept of completeness

The application allows a user to define any group of file types as a product. Once a product is defined, a user can view a summary of file time coverage for that product over an entire year. Through the Web client, a user may click on any individual day to view more detailed completeness information for that day.

Once a configured job is created, a user may view input and output completeness for that configured job over an entire year. Via the Web client, a user may click on any individual day to view more detailed input/output completeness information for that day.

6.4 Rich user interface

The HIRDLS SIPS application includes a rich, Web-based user interface. A navigation bar separates the view into several well-defined areas, such as distributions, products, processors, jobs, and deliveries.

A user may use the Web client to view the contents of text-based files or download files to his/her local machine. Additionally, users may use the Web interface to upload files directly into near line storage.

The Web client may also be used to select a time period over which to run a configured job. The Web client provides a feature for automatically finding the file instances to use as input to the job. Input file instances may also be queried for and selected manually. A user may ask to be notified via email once the job is complete.

The Web client also provides an interface for delivering files to the DAAC. After selecting an ESDT, an operator may then query for the desired file instances to deliver. Before submitting the delivery request to SIPS, the operator may ask to be notified via email when the delivery is complete.

6.5 Support for multiple client types

The HIRDLS SIPS application was designed to support multiple client types. Currently, two client types are supported. Java clients may communicate with the application through RMI. This is the most expansive interface to the application, and the presentation tier uses this interface to handle Web-based client connections.

Non-java clients may connect to the application through a more limited socket-based interface. Future work includes allowing a broader range of requests from clients using the socket-based interface.

6.6 Loose coupling

The HIRDLS SIPS application has a plug-and-play attitude towards the processors that it runs. New processors may be installed and configured at any time with little effort. The database persists the information needed to stage and run any installed processor in the future. With this approach, a particular SIPS installation may be configured to run virtually any processor.

Also, the HIRDLS SIPS application is capable of running on any operating system. Furthermore the application makes no restrictions as to the platform that the processors must run on.

Lastly, although this has been a discussion of the HIRDLS SIPS, the application itself has no HIRDLS-specific knowledge. The application may be used to process data for any instrument.

7. DISCUSSION

7.1 Supporting science processor development

SCF processors evolve with time and individual subsystems may be loosely coupled at first until the computational problem is completely understood. In addition there may be subsystems that exist only to perform certain internal consistency checks, perhaps to simulate data in the processing stream prior to receiving actual instrument data. HIRDLS SIPS is easily configured to handle the formative nature of processors. Subsystems can be linked as separate configurations of one or more executables. This allows processors to be gradually introduced into SIPS which benefits the science developer as user and system interfaces can be tested early in the project.

Another potential benefit of this ability to handle loosely coupled processor subsystems is to use SIPS as a computational steering system. Each step in a "processor" is configured as either a separate processor or separate configuration of the same processor. A job is then created linking the steps together into a virtual processor. The downside to this approach is that communications, other than diagnostics, is limited to file-to-file transfers. SIPS is never aware of processors science data output, so steps within the process must communicate with files. This requires SIPS to catalog the intermediate files.

7.2 Quality assessment

Quality assessment (QA) of operational processors and their data products is one portion of the overall validation process that HIRDLS is required to perform. Operational QA is also a major part of the feedback process that science processor developers can use to assess the performance and quality of the software.

SIPS does not necessarily know anything about the nature of the QA product, therefore a flexible one-to-many relationship mapping of QA products with their respective parent product is used. QA processors are only required to use a systematic approach to naming files, for instance, keywords separated by an underscore followed by a standard filename suffix (e.g., jpg, html, txt). If fields within the file can be mapped (using a regular expression) to search criteria, then SIPS can find files matching a particular search. For example, a search might be composed of: L2, March 2004, Ozone, 30km map. SIPS can match the dates and product using its own database and then look for regular expression matches for the remaining entries.

7.3 Future work

With the Aura launch scheduled for January 2004, one of the highest priority items is introducing automated processing. Another task is to expand the QA interface to make product and processor assessment easy and comprehensive. A powerful QA interface can provide the ability for extended science team members to assess quality via the Web GUI and potentially mark products as ready for delivery to the DAAC. Also, careful attention will be paid to making processor installation and configuration, job monitoring and other common tasks easier and more intuitive. Finally, extending the software application interface, which provides the SCF with a non-GUI interface to SIPS, will be on the to-do list. This interface will allow SCF developers to write custom post-processing scripts to find, retrieve and analyze products or information from SIPS.

ACKNOWLEDGEMENTS

We would like to thank Vince Dean for his help in testing SIPS and Charles Cavanaugh for the HIRDLS SIPS formative development. Special thanks go to Brendan Torpy for outstanding software development support. This work has been made possible through NASA contract NAS5-97046.

REFERENCES

1. McConnell, S., *Software Project Survival Guide*, Microsoft Press, Redmond WA, 1997.

2. McConnell, S., *Rapid Development*, Microsoft Press, Redmond WA, 1996.

3. Ramakrishnan, R. & Gehrke, J., *Database Management Systems, Second Edition*, McGraw-Hill Companies, Inc., 2000.

4. *Enterprise JavaBeans Technology,* http://java.sun.com/products/ejb

5. *Java Remote Method Invocation*, http://java.sun.com/products/jdk/rmi

6. Hall, M. & Brown, L., *Core Web Programming, Second Edition*, Sun Microsystems Press, New Jersey, 2001.

7. *Java Servlet Technology,* http://java.sun.com/products/servlet

8. *JavaServer Pages Technology,* http://java.sun.com/products/jsp

9. *Open Source Initiative,* http://www.opensource.org