

## 6. AITTL - Algorithm I&T CSCI

---

### 6.1 CSCI Overview

The purpose of the Algorithm Integration and Test Tools (AITTL) Computer Software Configuration Item (CSCI) is to provide the software tools required to integrate and test (I&T) the science software at the Distributed Active Archive Center (DAAC). The science software will be developed by a Science Computing Facility (SCF), which may be at a different location than the DAAC.

The division of responsibilities between the DAAC and the SCF is generally the following: The SCF is responsible for developing the science software and ensuring that the generated products are scientifically correct. The DAAC is responsible for integrating the science software into the production environment, ensuring that the software will run safely (i.e., will not interfere with the production environment or with other product generation), and running the software in a production mode.

The developer/operator division that is characteristic of the science software life cycle causes the DAAC I&T personnel to have certain special requirements. The I&T team needs to be able to do the following things:

- Receive a science software delivery from the SCF. The delivery will be made either electronically or via hard media.
- Examine the science software delivery for correctness and completeness. This includes: examining accompanying documentation, verifying that prescribed coding standards have been followed, and running preliminary static and dynamic diagnostic tools to check for potential errors. The delivered files must also be placed under configuration control.
- Compile and link the delivered source files.
- Run test cases. For the most part, these test cases will be supplied by the SCF as part of the delivery. Also supplied by the SCF, for each test case, will be a set of required input files and a corresponding set of output files. Since some of the input files may already reside at the DAAC, the I&T personnel also need the ability to manually stage inputs from the data servers. The DAAC I&T team will re-run each test case and compare their outputs with those supplied by the SCF. Because the SCF and DAAC machines may have different precision, the file comparison utility needs to be more sophisticated than the usual UNIX "diff" tool: it needs to be able to screen out differences that are due only to differences in precision. In addition, the ability to examine product metadata is required.
- Diagnose errors. This requires access to: interactive debuggers, screen dump utilities, data visualization tools, and so on.
- Collect resource requirement statistics. This includes: CPU time, memory requirements, disk space requirements, and so on. The collection of such statistics is required both as a "sanity check", to make sure that the measured requirements match the expected values, and also for the PGE Profile, which is used by the Planning and Processing systems to execute the science software properly.

- Update the system databases once the science software has completed acceptance testing. This includes: adding the source code and documentation to the data server, so that they may be distributed to requesting users, and adding the resource requirement information to the PGE Profile.
- Write reports and maintain the I&T log.
- Write additional ad-hoc test tools.

Satisfaction of these requirements is distributed across several systems. The ingest client for science software will be supplied by the Ingest Subsystem (INS) of the Science Data Processing Segment (SDPS). Configuration management and problem tracking tools will be provided by the Management Subsystem (MSS) of the Communications and System Management Segment (CSMS). Compilers, linkers, debuggers, and other development and operating system tools will be furnished by the Algorithm Integration and Test Hardware Configuration Item (AITHW) of the Data Processing Subsystem (DPS) of SDPS, since such utilities are closely wedded to the processing platforms. In addition, some of the standards checking and profiling requirements will probably be satisfied by AITHW as well, since certain of these capabilities will be found in compilers and development environments. The remaining requirements will be satisfied by AITTL.

The AITTL-supplied tools therefore fall into the following categories:

- Tools to view science software documentation.
- Tools to check compliance of science software to ESDIS-specified coding standards.
- Code analysis tools.
- Data visualization tools.
- HDF file comparison tool.
- Binary file comparison environment.
- Profiling tool.
- Tool to register the science software with the Planning and Data Processing system.
- Tools to add and update Science Software Archive Packages (SSAPs) in the Data Server.
- Tools for writing reports and maintaining the I&T logs.
- Tools for checking Process Control Files and for prohibited functions.
- Tools to display product metadata.

Most of the functions for the Algorithm Integration & Test CSCI are being developed as working prototypes and delivered for Interim Release 1 (Ir1). The Release B design approach is based on using Ir1 and any Release A changes as a basis for which to add Release B capabilities. Since AITTL CSCI is very operations oriented, as much feedback from DAAC Operations who have used the Ir1 AITTL tools will be incorporated into the Release B AITTL capabilities. Unlike Ir1, the Job Scheduler COTS product, AutoSys, will not be provided as part of AITTL to manage the execution of jobs on AITTL HW. To execute PGEs using AutoSys, the PGE must have a Profile defined in the PDPS database, and must be scheduled via the Planning Subsystem and executed via the Processing Subsystem. The tools to create and update a PGE Profile in the PDPS database are part of the AITTL CSCI and are described (along with the other functions of AITTL) in this section.

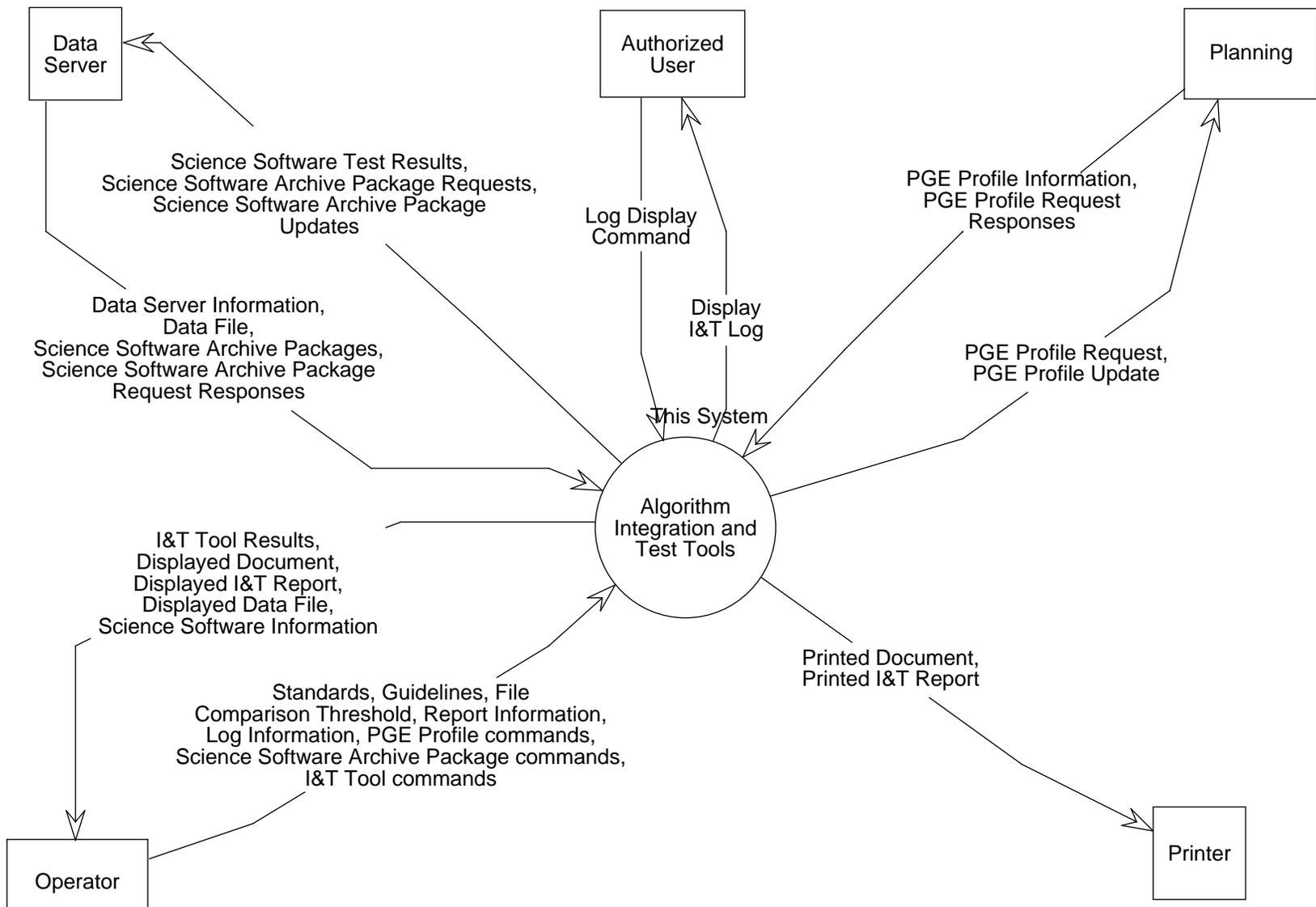
The formation of the AITTL design material is the following:

- a. Section 6.2—AITTL Context.
- b. Section 6.3—AITTL Object Model. This contains and describes the four object models of AITTL: AIT Manager GUI, Science Software Archive Package GUI, the PGE Registration GUI, and the manual interface to stage and destage files.
- c. Section 6.4—AITTL Class Descriptions. This describes the classes in the AITTL Object Models.
- d. Section 6.5—AITTL Dynamic Model. This has detailed event traces and descriptions for the classes in the AITTL Object Models.
- e. Section 6.6—AITTL Functional Model representing the many tool dependent data flows for AITTL.
- f. Section 6.7—AITTL Operational Scenarios, presents some operational scenarios showing how the AITTL tools may be used in the integration and test process, rather than the usual dynamic model.
- g. Section 6.8—AITTL Structure which defines the CSC components of the AITTL CSCI.
- h. Section 6.9—AITTL Management and Operation defines some management and operations concepts used in developing the AITTL CSCI design.

For additional information on science software integration and test procedures, see also: 205-CD-002-001, Science User's Guide and Operations Procedures Handbook for the ECS Project, Part 4, and JU9403V1, Science Software Integration and Test. For information on the ESDIS science software coding standards and guidelines, see: 423-16-01, Data Production Software and Science Computing Facility (SCF) Standards and Guidelines.

## **6.2 CSCI Context**

The context diagram for the AITTL CSCI is shown in Figure 6.2-1.



**Figure 6.2-1. AITTL Context Diagram**

AITTL has interfaces with two other ECS subsystems: the Data Server Subsystem and the Planning Subsystem.

The purpose of the interface with the Data Server is for AITTL to update and archive a tested science software delivery (*Science Software Archive Package*), and to archive test results (*Science Software Test Results*). The Data Server provides information about science software that is already archived on the data server (*Data Server Information*), transfers requested data files (*Data File*) and Science Software Archive Packages Requests (*Science Software Archive Package*), and issues responses to AITTL requests (*Science Software Archive Package Request Responses*).

The purpose of the interface with Planning is to update the PGE Profile (*PGE Profile Update*), part of the PDPS database that contains information defining each product generation executive (PGE) to the Planning and Data Processing Subsystems. This information is needed by Planning to effectively schedule and plan PGEs, and is passed along to the Data Processing Subsystem so that PGEs can be executed correctly and the results reported back to the operator. Information about PGEs already recorded in the PDPS database are requested from Planning (*PGE Profile Request*) and received from Planning (*PGE Profile Information*) along with the response (*PGE Profile Request Response*).

The primary interface for AITTL is with the operator(s) responsible for integration and test. The operator sends commands (*I&T Tool Commands*) to the various tools, supplies the standards checkers with the desired standards (*Standards*) and guidelines (*Guidelines*), supplies the file comparison utility with thresholds (*File Comparison Threshold*) to filter out precision differences, edits existing Science Software Archive Packages and creates new ones to link delivered source code with its test data, documentation, etc. (*Science Software Archive Package commands*), updates and creates the information about the tested PGE in the PDPS database (*PGE Profile commands*), and enters information into reports (*Report Information*) and into the integration and test log (*Log Information*). The tools display various results (*I&T Tool Results*), science software documentation (*Displayed Document*), Science Software Archive Package file lists and metadata and PGE Profile attributes (*Science Software Information*), and integration and test reports (*Displayed I&T Report*) for the operator to examine. The Authorized User is allowed to issue log commands (*Log Display Commands*) and receive a display of the I&T log (*Display I&T Log*). Finally, the tools send a hard copy of science software documentation (*Printed Document*), as well as integration and test reports and tool results (*Printed I&T Report*) to the printer.

### **6.3 CSCI Object Model**

The CSCI object model is broken into a number of views; Figures 6.3-1 to 6.3-4. Each view captures a different aspect of the CSCI capabilities. The AIT Manager object model is the starting point and spawns the classes in the other views via choices in its GUI menus. Complete descriptions of the classes are provided in text in Section 6.4.

### **6.3.1 AIT Manager GUI View**

This section gives the object model for the AIT Manager GUI. This GUI is used by the DAAC operator to check in and verify the science software code as delivered by the SCFs. The GUI runs instrument-specific compilation and execution scripts, configuration management scripts, custom code checking, file display and comparison tools, and COTS tools such as office automation and analysis environment programs. The AIT Manager GUI contains a graphical checklist of AI&T steps in delivery and testing of science software, and a display of a log file. All other views are started from this GUI via menu selections.

Figure 6.3-1 shows the Algorithm Integration and Test Manager Release B object model.



### **6.3.2 Science Software Archive Package (SSAP) GUI View**

This section gives the object model for the Science Software Archive Package (SSAP) GUI. This GUI is used by the DAAC operator(s) to view the list of existing SSAPs at the Data Server, to retrieve, modify, or delete SSAPs, or to create an entirely new SSAP and store it to the Data Server. This GUI spawns other GUIs (also shown on the diagram) that allow the operator to manipulate the list of files in an SSAP, or to view and alter the SSAP metadata. An access control class is provided to assure that any SSAP changes are authorized and logged.

Figure 6.3-2 shows the Science Software Archive Package (SSAP) GUI object model.

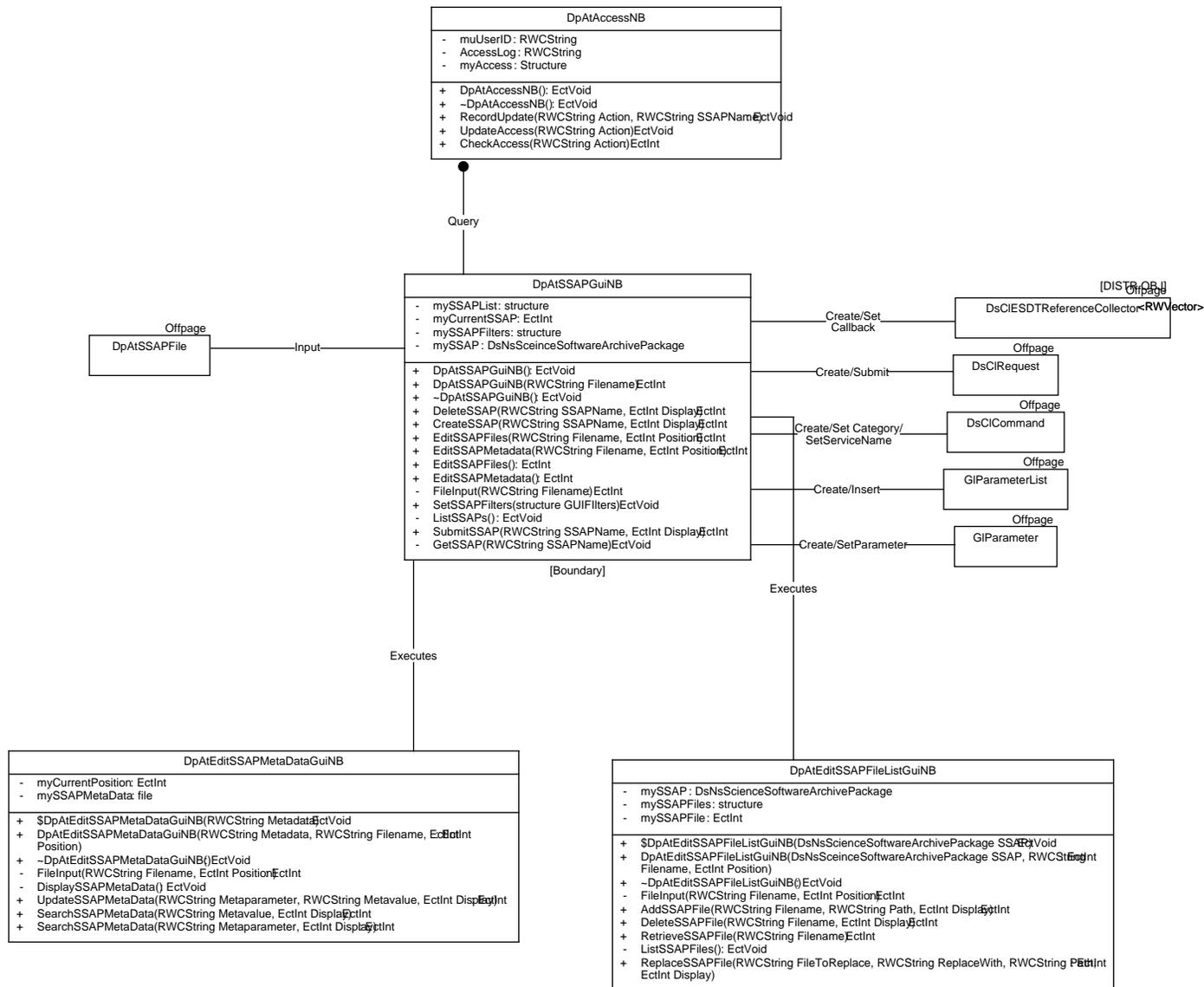


Figure 6.3-2. Science Software Archive Package GUI Object Model

### **6.3.3 PGE Registration GUI View**

This section gives the object model for the PGE Registration GUI. This GUI is used by the DAAC operator(s) to create the Profile for a new PGE, and to view and alter an existing PGE Profile in the PDPS database. The PGE Registration GUI allows the operator to alter/create the basic definition of a PGE in the PDPS database (including resource requirements and performance statistics) and creates the other GUIs (also shown on the diagram) that allow for the creation/altering of PGE user parameters, PGE inputs and outputs, and PGE activation rule definition.

Figure 6.3-3 shows the PGE Registration GUI object model.

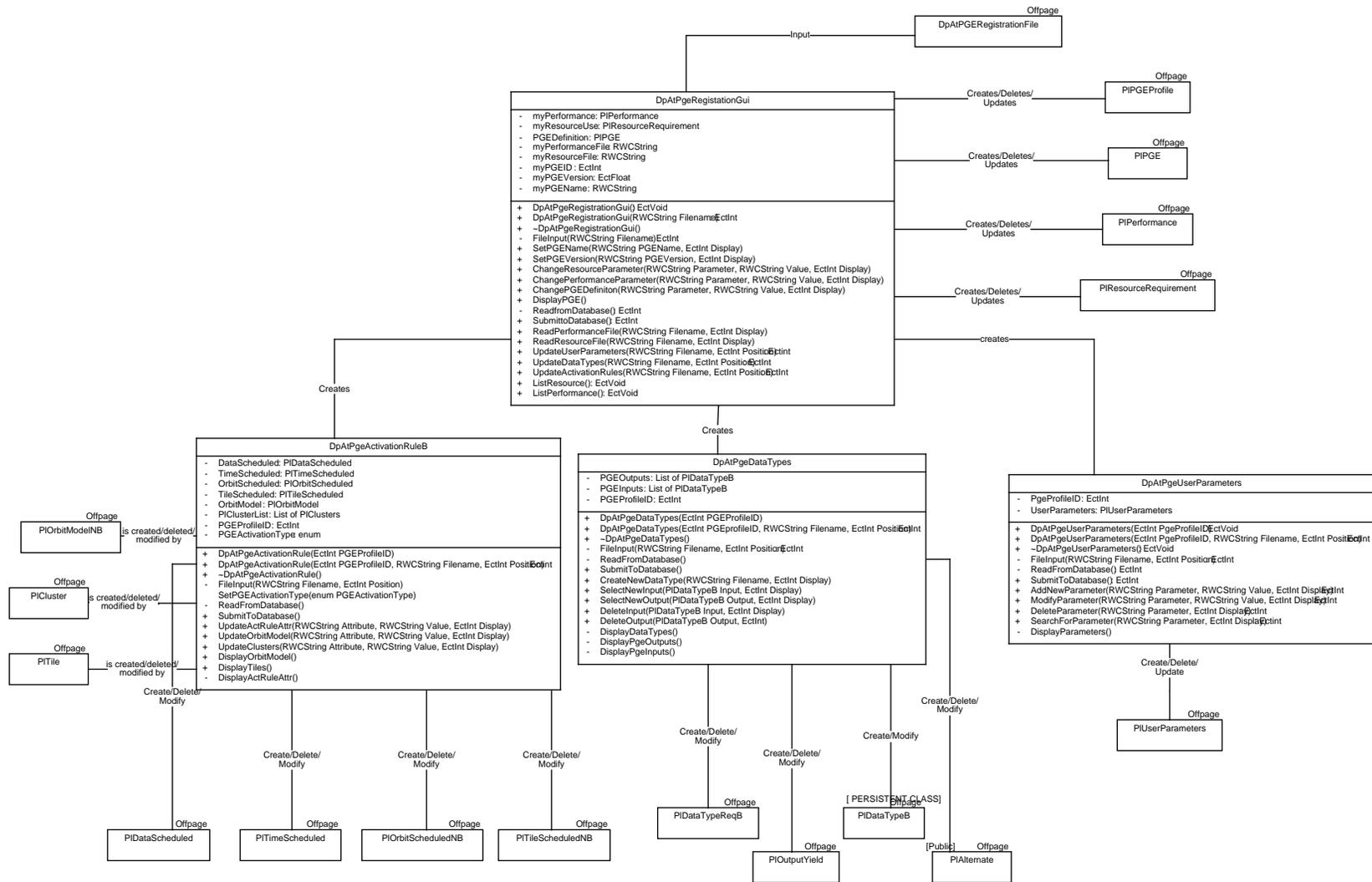
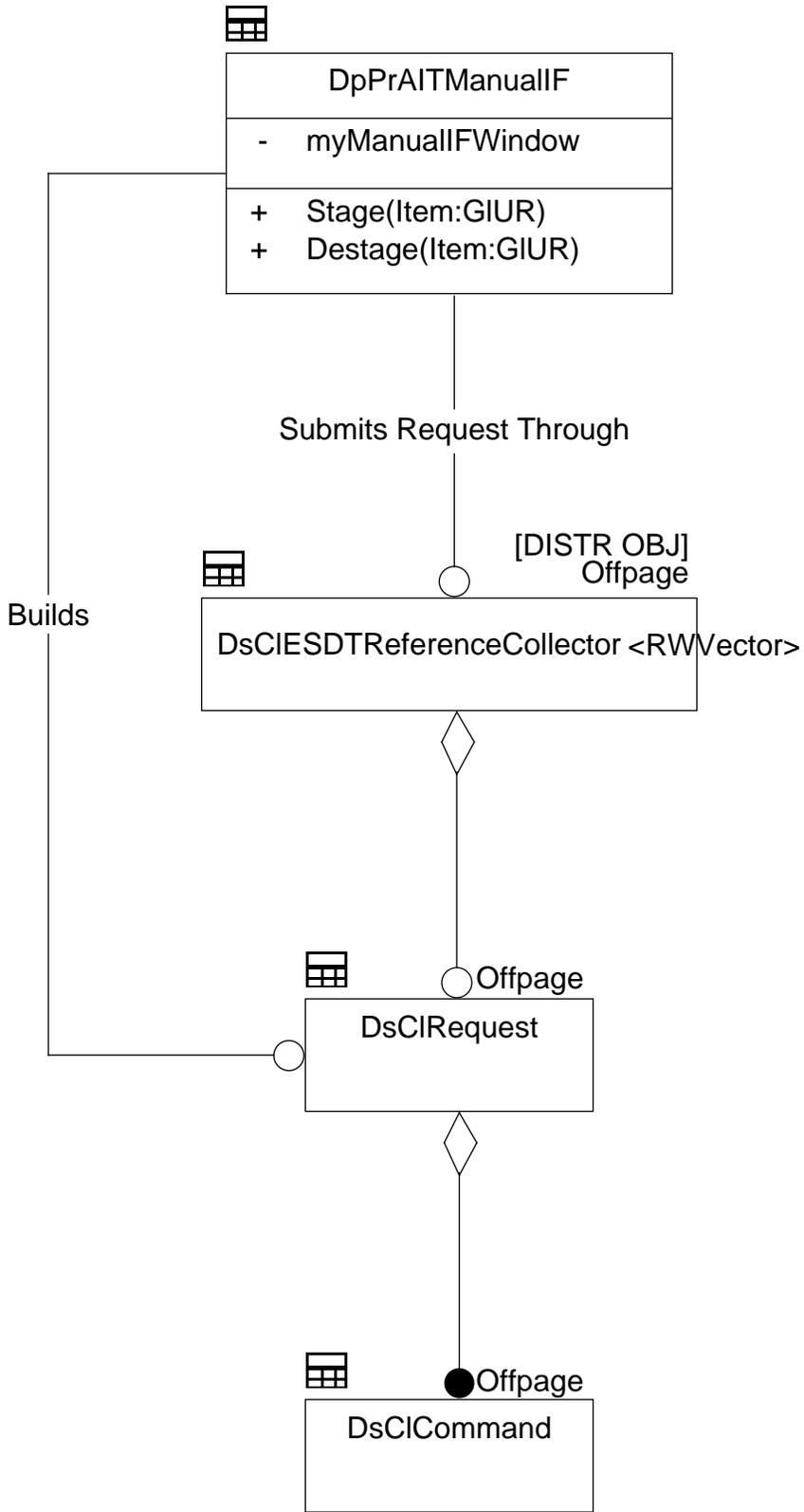


Figure 6.3-3. PGE Registration Object Model

### **6.3.4 Manual Staging and Destaging Tool View**

This section gives the object model for the manual staging and destaging GUI. This GUI/tool is used by the DAAC operator(s) to acquire or insert data to and from the Data Server during the Algorithm Integration and Test process.

Figure 6.3-4 shows the algorithm integration and test support for data sever I/F object model.



**Figure 6.3-4. Manual Staging and Destaging Interface**

## 6.4 CSCI Class Descriptions

### 6.4.1 Analysisenvironment Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

Run analysis environment. For Sun machines, SPARCWorks; for SGI machines, CASEVision. THIS IS NOT A CLASS. It is callable from the Unix command line.

#### Attributes:

None

#### Operations:

None

#### Associations:

The Analysisenvironment class has associations with the following classes:

Class: MgrGui SpawnProgram

### 6.4.2 CMscript Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

Script for use with CM tool (ClearCase) THIS IS NOT A CLASS. It is callable from the Unix command line.

#### Attributes:

None

#### Operations:

None

## Associations:

The CMscript class has associations with the following classes:

Class: MgrGui RunProgram

### 6.4.3 DpAtAccessNB Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class provides access control and an access log for the current user of the SSAP GUI. It provides routines to check the access given to a user, and allows for creation of and updates to the access log.

## Attributes:

**AccessLog** - This is the name + path of the log file created for this userid. It is created/updated by RecordUpdate.

Data Type:RWCString

Privilege:Private

Default Value:

**muUserID** - This is the user id of the current user. It will be retrieved via an MSS interface.

Data Type:RWCString

Privilege:Private

Default Value:

**myAccess** - This is the access permissions for the current user. It is a structure made up of a list of Actions (Create an SSAP, Delete an SSAP, Add File to SSAP File List, etc...) and whether the user is permitted to perform them.

Data Type:Structure

Privilege:Private

Default Value:

## Operations:

**CheckAccess** - This method returns the access permission of the user for the specified Action. The return is treated as a boolean.

Arguments:RWCString Action

Return Type:EctInt

Privilege:Public

PDL:

```
// Call MSS routine to get the current users UserID.  
// Return the access in myAccess for the Action.
```

**DpAtAccessNB** - This is the constructor of the class.

Arguments:

Return Type:EctVoid

Privilege:Public

```
PDL: // Call MSS routine to get the current users UserID.  
// Create a myAccess structure and initialize all permissions  
// to 0 (not allowed).
```

**RecordUpdate** - This routine updates the AccessLog file with the specified Action and the name of being manipulated. If the access log does not exist, this routine creates it.

Arguments:RWCString Action, RWCString SSAPName

Return Type:EctVoid

Privilege:Public

PDL:

```
// Call MSS routine to get the current users UserID.  
if (AccessLog = null)  
{  
// Create AccessLog file.  
}
```

```
// Add the current action and SSAPName to the file.
```

**UpdateAccess** - This method updates the user's access permission for the specified Action. It reverses the permission that existed before the call.

Arguments:RWCString Action

Return Type:EctVoid

Privilege:Public

PDL:

```
// Call MSS routine to get the current users UserID.  
// Toggle the Access for the entry in myAccess specified by Action.
```

**~DpAtAccessNB** - This is the destructor for the class.

Arguments:

Return Type:EctVoid

Privilege:Public

PDL:

```
// Call MSS routine to get the current users UserID.  
// Delete the myAccess structure.
```

Associations:

The DpAtAccessNB class has associations with the following classes:

Class: DpAtSSAPGuiNB Query

#### 6.4.4 DpAtEditSSAPFileListGuiNB Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class provides a GUI definition for adding & deleting files from the specified SSAP. It also allows for the extraction of the chosen file from the SSAP to the local directory.

##### Attributes:

**mySSAP** - This is the Science Software Archive Package from which the file list is extracted.

Data Type:DsNsScienceSoftwareArchivePackage

Privilege:Private

Default Value:

**mySSAPFile** - This is the currently selected file in the SSAP. It is an index into the list of files in the SSAP.

Data Type:EctInt

Privilege:Private

Default Value:

**mySSAPFiles** - This is a list of the files in a SSAP, grouped according to types and functions.

Data Type:structure

Privilege:Private

Default Value:

##### Operations:

**\$DpAtEditSSAPFileListGuiNB** - This is the constructor for the class. It creates the Edit SSAP File List GUI, and creates myFileList from the provided SSAP.

Arguments:DsNsScienceSoftwareArchivePackage SSAP

Return Type:EctVoid

Privilege:Public

PDL:

```
// Create the EditSSAPFileListGui through
```

```

// Builder Xcessory generated code.

// mySSAP is initalized with the value passed in.
mySSAP=SSAP;

// mySSAPFiles is created from the SSAP record passed in.
// mySSAPFile is initialized to point to the first file in the
// list.

// ListSSAPFiles is called to list the files in the SSAP to the screen.
ListSSAPFiles ();

```

**AddSSAPFile** - This method adds the file specified by Filename and Path to the SSAP. If Display is true, the change is displayed to the screen by calling ListSSAPFiles

Arguments:RWCString Filename, RWCString Path, EctInt Display

Return Type:EctInt

Privilege:Public

PDL:

```

// mySSAPFiles is searched for the Filename to be added.

if (the Filename does not already exist)
{
// Call CheckAccess to verify that the user is permitted to
// add files to an SSAP. If the user does not have permission
// to add files, and Display is true, a message is sent to the
// screen to indicate that the action is not allowed.

// The Filename specified is added to mySSAPFiles.

// The file specified by Filename and Path is inserted into
// mySSAP.

// RecordUpdate is called to update the log to the fact that
// the current user added a file to the SSAP.
}
else
{
if Display

// The user is prompted as to if the user wants to overwrite
// the file.

if (user accepts overwrite)
{
// Call CheckAccess to verify that the user is permitted to

```

```

// replace files in an SSAP. If the user does not have permission
// to replace files, and Display is true, a message is sent to the
// screen to indicate that the action is not allowed.

// The file specified by Filename and Path is inserted into
// mySSAP over the old version of the file.

// RecordUpdate is called to update the log to the fact that
// the current user replaced a file to the SSAP.
}
}

```

```

if Display
// ListSSAPFiles is called to display the list of files in the SSAP.
ListSSAPFiles ();

```

**DeleteSSAPFile** - This method deletes the file specified by Filename from the SSAP. If Display is true, the change is displayed to the user via a call to ListSSAPFiles.

Arguments:RWCString Filename, EctInt Display

Return Type:EctInt

Privilege:Public

PDL:

```

// mySSAPFiles is searched for the specified Filename.

if (Filename found)
{

// Call CheckAccess to verify that the user is permitted to
// delete files from an SSAP. If the user does not have permission
// to delete files, and Display is true, a message is sent to the
// screen to indicate that the action is not allowed.

// The Filename specified is deleted from mySSAPFiles.

// The file specified by Filename is deleted from
// mySSAP.

// RecordUpdate is called to update the log to the fact that
// the current user deleted a file from the SSAP.

if Display
// ListSSAPFiles is called to display the list of files in the SSAP.
ListSSAPFiles ();
}
}

```

**DpAtEditSSAPFileListGuiNB** - This constructor takes a Filename and Position as inputs in addition to the SSAP to be modified. It calls routines within this class to parse the input file starting at the specified position, and make any modifications to the file list specified in the input file.

Arguments: DsNsSceinceSoftwareArchivePackage SSAP, RWCString Filename, EctInt Position

Return Type: EctInt

Privilege: Public

PDL:

```
// mySSAP is initialized with the value passed in.
mySSAP=SSAP;

// mySSAPFiles is created from the SSAP record passed in.
// mySSAPFile is initialized to point to the first file in the
// list.

// FileInput is called to parse the input file from position
// for file list updates.
FileInput (Filename, Position);
```

**FileInput** - This method parses the file defined by Filename from the Position specified. It calls routines within this class to perform the SSAP file list changes defined in the file.

Arguments: RWCString Filename, EctInt Position

Return Type: EctInt

Privilege: Private

PDL:

```
// Open the file specified by Filename.

while (not at Position in the file)
// Read the next line in the file.

while (there is more to read in the file)
{
// Parse each line in the file.
switch
{
// For each type of file input command, call the
// appropriate routine within this class to
// process it.

if (bad input from file)
// Exit loop.

if (bad return code from one of the called routines)
```

```

        // Exit Loop.

        if (input command is for Metadata or SSAP)
            // Exit loop.
        }
    }

    // Close the file.
    // return status(if negative) or position in the file.

```

**ListSSAPFiles** - This routine displays the list of files in the SSAP on the GUI.

Arguments:

Return Type:EctVoid

Privilege:Private

PDL: while (not done with the list)

```

    {
        // Display the name of the file in position i of
        // mySSAPFiles.
    }

```

**ReplaceSSAPFile** - This method replaces the file in the SSAP defined by FileToReplace with the file defined by ReplaceWith and Path. If Display is true, the change is displayed to the GUI via a call to ListSSAPFiles.

Arguments:RWCString FileToReplace, RWCString ReplaceWith, RWCString Path, EctInt Display

Return Type:EctInt

Privilege:Public

PDL:

```

    // Open the file specified by Filename.

```

```

while (not at Position in the file)
    // Read the next line in the file.

```

```

while (there is more to read in the file)

```

```

{
    // Parse each line in the file.
    switch
    {
        // For each type of file input command, call the
        // appropriate routine within this class to
        // process it.

```

```

        if (bad input from file)

```

```

    // Exit loop.

    if (bad return code from one of the called routines)
        // Exit Loop.

    if (input command is for Metadata or SSAP)
        // Exit loop.
    }
}

// Close the file.
// return status(if negative) or position in the file.

```

**RetrieveSSAPFile** - This method retrieves the file specified by Filename from the SSAP and places it in the local directory.

Arguments:RWCString Filename

Return Type:EctInt

Privilege:Public

PDL:

```

// Initalize Data Server objects.
// Create a Data Server command to ACQUIRE the file specified
// by Filename. Send the command to the Data Server.
// The file is placed in the local directory.

```

**~DpAtEditSSAPFileListGuiNB** - This is the destructor for the class. It removes the GUI from the display.

Arguments:

Return Type:EctVoid

Privilege:Public

PDL:

```

// Remove the EditSSAPFileListGui from the
// display using Builder Xcessory generated code.

```

Associations:

The DpAtEditSSAPFileListGuiNB class has associations with the following classes:

Class: DpAtSSAPGuiNB Executes

## 6.4.5 DpAtEditSSAPMetaDataGuiNB Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class defines the GUI that provides the ability to update the metadata of the selected SSAP. It displays the metadata file passed as an argument to the constructor, and provides the capability to update and search the SSAP metadata.

### Attributes:

**myCurrentPosition** - This is the current position in the metadata. It is set by the search methods, as well as moving the cursor on the GUI.

Data Type:EctInt

Privilege:Private

Default Value:

**mySSAPMetaData** - This is the metadata file associated with the SSAP. Note that all metadata is in parameter=value format, so the file has a listing of each metadata field with its corresponding value.

Data Type:file

Privilege:Private

Default Value:

### Operations:

**\$DpAtEditSSAPMetaDataGuiNB** - This is the constructor for the class. It creates the Edit SSAP Metadata GUI, and displays the SSAP's metadata via a call to DisplaySSAPMetaData.

Arguments:RWCString Metadata

Return Type:EctVoid

Privilege:Public

PDL:

```
// Create the EditSSAPMetaDataGui through  
// Builder Xcessory generated code.
```

```
// mySSAPMetaData is initalized with the value passed in.  
mySSAPMetaData=Metadata;
```

```
// DisplaySSAPMetadata called to show the metadata  
// on the user's screen.
```

```
DisplaySSAPMetadata ();
```

**DisplaySSAPMetaData** - This method displays the metadata to the screen in the parameter=value format.

Arguments:

Return Type:EctVoid

Privilege:Private

PDL:

```
while (not end of metadata)
{
  // Display metadata in parameter=value format
}
```

**DpAtEditSSAPMetaDataGuiNB** - This constructor takes in a Filename and Position in addition to the SSAP metadata and calls FileInput to parse the specified file.

Arguments:RWCString Metadata, RWCString Filename, EctInt Position

Return Type:EctInt

Privilege:Public

PDL:

```
// mySSAPMetaData is initialized with the value passed in.
mySSAPMetaData=Metadata;

// FileInput is called to continue parsing the input file
// for metadata updates.
FileInput (Filename, Position);
```

**FileInput** - This method parses the file specified by Filename from the argument Position. It calls the routines of this class to make the updates to the metadata defined in the file.

Arguments:RWCString Filename, EctInt Position

Return Type:EctInt

Privilege:Private

PDL:

```
// Open the file specified by Filename.

while (not to Position in the file)
  // Read the next line.

while (there is more to read in the file)
{
  // Parse the next line in the file.
  switch
  {
    // For each type of file input command, call the
```

```

// appropriate routine within this class to
// process it.

if (bad input from file)
    // Exit loop.

if (bad return code from called routine)
    // Exit loop.

if (input command is of type FileList or SSAP)
    // Exit loop.
}
}

// Close the file.
// return current position or "bad" status.

```

**SearchSSAPMetaData** - This method searches the metadata for the specified value. If found, it sets myCurrentPosition to the line of the Value. If Display is true it also sets the GUI to the line where the Value was found.

Arguments:RWCString Metavalue, EctInt Display

Return Type:EctInt

Privilege:Public

PDL:

```

while (not end of metadata)
{
    if (current metadata value = Metavalue)
        // Set myCurrentPosition to the position of the
        // found value.
        if Display
            // Highlight found metadata line on the display.
}

if (value not found)
    // Set myCurrentPosition to beyond the last entry in the file.

```

**SearchSSAPMetaData** - This method searches the metadata for the specified Parameter. if found, it sets myCurrentPosition to the location of the Parameter in the file. If Display is true, it also sets the GUI to the found Parameter.

Arguments:RWCString Metaparameter, EctInt Display

Return Type:EctInt

Privilege:Public

PDL:

```

while (not end of metadata)
{
  if (current metadata parameter = Metaparameter)
    // Set myCurrentPosition to the position of the
    // found parameter.
    if Display
      // Highlight found metadata line on the display.
}

if (value not found)
  // Set myCurrentPosition to beyond the last entry in the file.

```

**UpdateSSAPMetaData** - This routine updates the specified Metaparameter with the specified Metavalue. If the Metaparameter cannot be found, it queries the user if he/she wishes to add the parameter to the file. If Display is true, DisplaySSAPMetaData is called to display the changes.

Arguments:RWCString Metaparameter, RWCString Metavalue, EctInt Display

Return Type:EctInt

Privilege:Public

PDL:

```

if (metadata parameter at myCurrentPosition = Metaparamter)
{
  // Call CheckAccess to detemine if the user has access to update
  // the metadata of the SSAP. If the return indicates that the
  // updating of metadata is not permitted, and Display is true,
  // a message is displayed indicating that the selected action
  // is not allowed.

  // Set the value at myCurrentPosition to Metavalue.

  // Call RecordUpdate to log the fact that the user updated
  // the metadata of the SSAP.
}
else
{
  // Call SearchSSAPMetaData to search for the specified parameter.
  SearchSSAPMetaData (Metaparameter, Display);

  if (metadata parameter at myCurrentPosition = Metaparamter)
  {
    // Call CheckAccess to detemine if the user has access to update
    // the metadata of the SSAP. If the return indicates that the
    // updating of metadata is not permitted, and Display is true,
    // a message is displayed indicating that the selected action
    // is not allowed.
  }
}

```

```

// Set the value at myCurrentPosition to Metavalue.

// Call RecordUpdate to log the fact that the user updated
// the metadata of the SSAP.
}
else
{
// Call CheckAccess to determine if the user has access to update
// the metadata of the SSAP. If the return indicates that the
// updating of metadata is not permitted, and Display is true,
// a message is displayed indicating that the selected action
// is not allowed.

// Add Metaparamter and Metavalue at myCurrentPosition
// which is the end of the Metadata file.

// Call RecordUpdate to log the fact that the user updated
// the metadata of the SSAP.
}
}

```

**~DpAtEditSSAPMetaDataGuiNB** - This is the destructor for the class. It is called when the GUI is exited or when the File Input is complete.

Arguments:

Return Type: EctVoid

Privilege: Public

PDL:

```

// Remove the EditSSAPMetaDataGui from the
// display using Builder Xcessory generated code.

```

Associations:

The DpAtEditSSAPMetaDataGuiNB class has associations with the following classes:

Class: DpAtSSAPGuiNB Executes

#### 6.4.6 DpAtMgr Class

Parent Class: Not Applicable

Public: No

Distributed Object: No

Purpose and Description:

General processor of AIT Manager Kicks off COTS, custom and instrument-specific

scripts Calls checklist, log, other processors

### Attributes:

**myHelpFileLogicals** - This is the pointer to the information for the help menu.

Data Type:PGSt\_PC\_Logical\*

Privilege:Private

Default Value:

### Operations:

**Checklist** - Checklist processor Called by DpAtMgr.Processor() Input: DpAtMgrChecklistData instance

Arguments:DpAtMgrChecklistData,EcTBoolean\*

Return Type:EcTInt

Privilege:Public

**File** - File menu processor Called by DpAtMgr.Processor() Input: menuSelection, ...

Arguments:EcTInt\*

Return Type:EcTInt

Privilege:Public

**Help** - Help menu processor

Arguments:EcTInt\*

Return Type:EcTInt

Privilege:Public

**Log** - Log processor Called by DpAtMgr.Processor()

Arguments:DpAtMgrLogData,DpAtMgrChecklistData

**Options** - Options menu processor Called by DpAtMgr.Processor() Input: menuSelection,

...

Arguments:EcTInt\*

Return Type:EcTInt

Privilege:Public

**Processor** - Main processor for data returned by GUI

Arguments:DpAtMgrGuiActivityData, DpAtMgrChecklistData, DpAtMgrLogData

Return Type:EcTInt

Privilege:Public

**Associations:**

The DpAtMgr class has associations with the following classes:

Class: DpAtMgrChecklistData  
GetActivityFlagSaveToFileChangeItemStateCurrentIndexIsChecked  
Class: DpAtMgrGuiActivityData  
GetSelectedAreaGetMenuSelectionPutActivityRequestGetProgramReturn Value  
Class: DpAtMgrCom Processorctor  
Class: DpAtMgrLogData  
PutActivityFlagGetActivityFlagWriteLogEntryReadLogEntryFindLogEntryGuiEditLog  
Annotation

**6.4.7 DpAtMgrBinaryFileEnvironmentGui Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class represents the definition of the Binary File Environment GUI.

**Attributes:**

None

**Operations:**

None

**Associations:**

The DpAtMgrBinaryFileEnvironmentGui class has associations with the following classes:

Class: DpAtMgrCheckPcfGui executes  
Class: DpAtMgrCheckProhibFuncGui executes  
Class: MgrGui executes

**6.4.8 DpAtMgrCheckHdfFile Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

IDL program to compare two HDF files, and also display metadata. THIS IS NOT A CLASS. It is callable from the Unix command line.

**Attributes:**

None

**Operations:**

None

**Associations:**

The DpAtMgrCheckHdfFile class has associations with the following classes:

Class: MgrGui RunProgram

#### **6.4.9 DpAtMgrCheckPcfGui Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

DpAtMgrCheckPcfGui is the GUI for checking Process Control Files (PCFs) for valid syntax and required contents.

**Attributes:**

None

**Operations:**

None

**Associations:**

The DpAtMgrCheckPcfGui class has associations with the following classes:

Class: DpAtMgrBinaryFileEnvironmentGui executes

#### 6.4.10 DpAtMgrCheckProhibFuncCom Class

Parent Class:Not Applicable  
Public:No  
Distributed Object:No  
Purpose and Description:  
Prohibited function checker, Unix command line version

##### **Attributes:**

None

##### **Operations:**

None

##### **Associations:**

The DpAtMgrCheckProhibFuncCom class has associations with the following classes:

Class: DpAtMgrCheckProhibFuncGui

Class: DpAtMgrProhibFuncListData

#### 6.4.11 DpAtMgrCheckProhibFuncGui Class

Parent Class:Not Applicable  
Public:No  
Distributed Object:No  
Purpose and Description:  
Input GUI for prohibited function checker

##### **Attributes:**

None

##### **Operations:**

None

### Associations:

The DpAtMgrCheckProhibFuncGui class has associations with the following classes:

Class: DpAtMgrCheckProhibFuncCom

Class: DpAtMgrBinaryFileEnvironmentGui executes

## 6.4.12 DpAtMgrChecklistData Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

Stores data for AIT Manager checklist

### Attributes:

**myActivityFlag** - Flag to indicate activity received from GUI =0, No activity =1, myCurrentIndex changed state selected/not selected =2, "Save Checklist" button pushed

Data Type:EcInt

Privilege:Private

Default Value:

**myCurrentIndex** - Current index of checklist item If myActivityFlag=1 received from GUI, this item changed state from/to checked/unchecked

Data Type:EcInt

Privilege:Private

Default Value:

**myItemIds** - Array of checklist identifiers, numbered consecutively One for each checklist item

Data Type:EcInt

Privilege:Private

Default Value:

**myItemIsChecked** - Array of states for checklist items =EcDFalse, item is not checked =EcDTrue, item is checked

Data Type:EcInt

Privilege:Private

Default Value:

**myLabels** - Array of checklist labels to display on GUI screen One label for each checklist item

Data Type:RWCString

Privilege:Private

Default Value:

**myNumItems** - Number of items for this checklist

Data Type:EctInt

Privilege:Private

Default Value:

### Operations:

**\$DpAtMgrChecklistData** - Constructor for class DpAtMgrChecklist data Reads checklist data from file

Arguments:

Return Type:Void

Privilege:Public

**ChangeItemState** - Changes state myItemIsChecked[ myCurrentIndex ] from/to checked/unchecked

Arguments:

Return Type:Void

Privilege:Public

**CurrentIndexIsChecked** - Returns EcDTrue if myCurrentIndex is checked; EcDFalse if not

Arguments:

Return Type:Void

Privilege:Public

**GetActivityFlag** - Get value of myActivityFlag

Arguments:

Return Type:Void

Privilege:Public

**GetCurrentIndex** - Gets index (ID) of current checklist item

Arguments:

Return Type:Void

Privilege:Public

**PutActivityFlag** - Set value of myActivityFlag

Arguments:EcTInt  
Return Type:Void  
Privilege:Public

**PutCurrentIndex** - Sets index (ID) of current checklist item

Arguments:EcTInt  
Return Type:Void  
Privilege:Public

**SaveToFile** - Saves checklist data to a file

Arguments:  
Return Type:Void  
Privilege:Public

### **Associations:**

The DpAtMgrChecklistData class has associations with the following classes:

Class: DpAtMgr GetActivityFlagSaveToFileChangeItemStateCurrentIndexIsChecked

Class: DpAtMgrInstrConfigData GetChecklistFileLogical

Class: MgrGui PutActivityFlagPutCurrentIndex

Class: DpAtMgrCom ctor

### **6.4.13 DpAtMgrCmdLineData Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

Stores all data which may be specified on the initial AIT Manager command line.

### **Attributes:**

**myInstrConfigLogical** - Logical ID in Process Control File (PCF) for instrument configuration data for this AIT Manager session

Data Type:EcTInt

Privilege:Private

Default Value:

**myInstrumentName** - Name of instrument for this AIT Manager session

Data Type:RWCString

Privilege:Private

Default Value:

**myStaticMotifRcFileLogical** - PCF file logical for user-editable static Motif menu data for this AIT Manager session \*\*\*\*\*This file is in the same format as the Motif resources file It contains all menu labels, sub-menu labels, etc. \*except\* the Run menu data, plus any user preferences such as colrs, fonts, etc.

Data Type:RWCString

Privilege:Private

Default Value:

## Operations:

**\$DpAtMgrCmdLineData** - Constructor for DpAtMgrCmdLineData

Arguments:

Return Type:Void

Privilege:Public

**GetInstrConfigLogical** - Gets instrument configuration file logical ID for PCF

Arguments:

Return Type:Void

Privilege:Public

**GetInstrumentName** - Gets instrument name

Arguments:

Return Type:Void

Privilege:Public

**GetStaticMotifRcFileLogical** - Gets static motif resources file logical for PCF

Arguments:

Return Type:Void

Privilege:Public

**PutInstrConfigLogical** - This outputs the value of the attribute myInstrConfigLogical.

Arguments:PGSt\_PC\_Logical

Return Type:Void

Privilege:Public

**PutInstrumentName** - Puts instrument name into storage

Arguments:EcTChar\*

Return Type:Void

Privilege:Public

**PutStaticMotifRcFileLogical** - This method outputs the value in the attribute my StaticMotifRcFileLogical.

Arguments:PGSt\_PC\_Logical

Return Type:Void

Privilege:Public

**WriteToFile** - Writes command line configuration for this AIT Manager session to file

Arguments:

Return Type:Void

Privilege:Public

#### **Associations:**

The DpAtMgrCmdLineData class has associations with the following classes:

Class: DpAtMgrInstrConfigData GetInstrConfigLogicalGetStaticMotifRcFileLogical

Class: DpAtMgrCom ctor

#### **6.4.14 DpAtMgrCom Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

Main program module for invoking AIT Manager. THIS IS NOT A CLASS. It is the main program, callable from the Unix command line.

#### **Attributes:**

None

#### **Operations:**

None

#### **Associations:**

The DpAtMgrCom class has associations with the following classes:

Class: MgrGui Processorctor()

Class: DpAtMgr Processorctor

Class: DpAtMgrChecklistData ctor  
Class: DpAtMgrCmdLineData ctor  
Class: DpAtMgrGuiActivityData ctor  
Class: DpAtMgrInstrConfigData ctor  
Class: DpAtMgrLogData ctor

#### 6.4.15 DpAtMgrGuiActivityData Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class is the interface between the GUI/Motif code and the external code. GUI callbacks are accesses of this class's data through its operations.

##### Attributes:

**myActivityRequest** - GUI activity requested by calling module =0, take no action =1, redisplay entire GUI =2, redisplay checklist only =3, redisplay log only =4, destroy entire GUI =5, edit current log entry

Data Type:EcTInt

Privilege:Private

Default Value:

**myMenuSelection** - Index of item selected on menus myMenuSelection[0] = main menu item index myMenuSelection[0]=0, File menu myMenuSelection[0]=1, Options menu myMenuSelection[0]=2, Tools menu myMenuSelection[0]=3, Run menu myMenuSelection[0]=4, Utilities menu myMenuSelection[0]=5, Help menu myMenuSelection[1] = sub menu item index myMenuSelection[1]=0, 1st sub menu item etc myMenuSelection[2] = sub sub menu item index myMenuSelection[2]=0, 1st sub sub menu item

Data Type:array of EcTInt

Privilege:Private

Default Value:

**myProgramReturnValue** - Return value from program selected from Tools, Run or Help menu

Data Type:EcTInt

Privilege:Private

Default Value:

**mySelectedArea** - Portion of GUI that the user clicked on =0, no selection =1, main menu

=2, checklist =3, log

Data Type:EcTInt

Privilege:Private

Default Value:

**myText** - Text written by user into GUI Used to annotate log

Data Type:RWCStringg

Privilege:Private

Default Value:

## Operations:

**\$DpAtMgrGuiActivityData** - Constructor for DpAtMgrGuiActivityData

Arguments:

Return Type:Void

Privilege:Public

**GetActivityRequest** - Get value of myActivityRequest

Arguments:

Return Type:Void

Privilege:Public

**GetMenuSelection** - Get value of myMenuSelection

Arguments:

Return Type:Void

Privilege:Public

**GetProgramReturnValue** - Get value of myProgramReturnValue

Arguments:

Return Type:Void

Privilege:Public

**GetSelectedArea** - Get value of mySelectedArea

Arguments:

Return Type:Void

Privilege:Public

**PutActivityRequest** - Set value of myActivityRequest

Arguments:EcTInt

Return Type:Void

Privilege:Public

**PutMenuSelection** - Set value of myMenuSelection

Arguments:EcTInt\*  
Return Type:Void  
Privilege:Public

**PutProgramReturnValue** - Set value of myProgramReturnValue

Arguments:EcTInt  
Return Type:Void  
Privilege:Public

**PutSelectedArea** - Set value of mySelectedArea

Arguments:EcTInt  
Return Type:Void  
Privilege:Public

### Associations:

The DpAtMgrGuiActivityData class has associations with the following classes:

Class: DpAtMgr  
GetSelectedAreaGetMenuSelectionPutActivityRequestGetProgramReturnValue  
Class: MgrGui  
PutSelectedAreaPutMenuSelectionGetActivityRequestPutProgramReturnValue  
Class: DpAtMgrCom ctor

### 6.4.16 DpAtMgrInstrConfigData Class

Parent Class:Not Applicable  
Public:No  
Distributed Object:No  
Purpose and Description:  
Stores instrument-specific configuration data

### Attributes:

**myChecklistFileLogical** - Checklist file PCF logical for this instrument configuration

Data Type:PGSt\_PC\_Logical  
Privilege:Private  
Default Value:0

**myLogFileLogical** - Log file PCF logical for this instrument configuration

Data Type:PGSt\_PC\_Logical

Privilege:Private  
Default Value:0

**myNumScripts** - Number of scripts available for this instrument configuration

Data Type:EcTInt  
Privilege:Private  
Default Value:0

**myScriptFileLogicals** - Script file PCF logicals for this instrument configuration, one for each script

Data Type:PGSt\_PC\_Logical\*  
Privilege:Private  
Default Value:

**myScriptLabels** - Script labels for this instrument configuration, for display on GUI menu, one for each script

Data Type:EcTChar\*\*  
Privilege:Private  
Default Value:

**myScriptOptions** - Command line options available for this instrument configuration, one for each script

Data Type:EcTChar\*\*  
Privilege:Private  
Default Value:

## Operations:

**\$DpAtMgrInstrConfigData** - Constructor for DpAtMgrInstrConfigData

Arguments:  
Return Type:Void  
Privilege:Public

**GetChecklistFileLogical** - Gets myChecklistFileLogical

Arguments:  
Return Type:PGSt\_PC\_Logical  
Privilege:Public

**GetLogFileLogical** - Gets myLogFileLogical

Arguments:  
Return Type:PGSt\_PC\_Logical  
Privilege:Public

**ReadFile** - Read an instrument configuration file to set all the private data members of this class

Arguments:

Return Type:EcTInt

Privilege:Public

**WriteMotifRcFile** - Reads the static Motif resources file, and writes its data and the Run/script data to the dynamic Motif resources file

Arguments:

Return Type:EcTInt

Privilege:Public

**WriteToFile** - Write all private data members of this class to an instrument configuration file

Arguments:

Return Type:EcTInt

Privilege:Public

#### **Associations:**

The DpAtMgrInstrConfigData class has associations with the following classes:

Class: DpAtMgrChecklistData GetChecklistFileLogical

Class: DpAtMgrCmdLineData GetInstrConfigLogicalGetStaticMotifRcFileLogical

Class: DpAtMgrLogData GetLogFileLogical

Class: DpAtMgrCom ctor

#### **6.4.17 DpAtMgrLogData Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

Stores data for AIT Manager instrument-specific log

#### **Attributes:**

**myActivityFlag** - Flag to indicate activity received from GUI =0, No activity =1, Checklist item was checked =2, "NEXT" button pushed =3, "CANCEL" button pushed =4, "LAST" button pushed

Data Type:EcTInt

Privilege:Private  
Default Value:0

**myAnnotation** - Annotation text for this log entry

Data Type:EcTChar\*  
Privilege:Private  
Default Value:

**myChecklistIndex** - Index of checklist item which generated this log entry

Data Type:EcTInt  
Privilege:Private  
Default Value:0

**myDate** - Date of this log entry

Data Type:DpTAtMgrDate  
Privilege:Private  
Default Value:

**myFileHandle** - Log file handle for input to read/write functions

Data Type:PGSt\_IO\_Gen\_FileHandle  
Privilege:Private  
Default Value:

**myItemId** - Unique log entry ID

Data Type:EcTInt  
Privilege:Private  
Default Value:0

**myTime** - Time of this log entry

Data Type:DPTAtMgrTime  
Privilege:Private  
Default Value:

## Operations:

**\$DpAtMgrLogData** - DpAtMgrLogData constructor Reads last entry in log file

Arguments:  
Return Type:Void  
Privilege:Public

**EditLogAnnotation** - Edits or creates a log entry annotation in a text editor window

Arguments:  
Return Type:EcTInt

Privilege:Public

**FindLogEntryGui** - GUI for searching for a text string in the log file entries (not the annotations)

Arguments:

Return Type:EcTInt

Privilege:Public

**GetActivityFlag** - Get value of myActivity flag

Arguments:

Return Type:EcTInt

Privilege:Public

**PutActivityFlag** - This method outputs the value in myActivityFlag.

Arguments:EcTInt

Return Type:EcTInt

Privilege:Public

**PutChecklistIndex** - Sets value of myChecklistIndex

Arguments:EcTInt

Return Type:Void

Privilege:Public

**ReadLogEntry** - Read a log entry from the log file

Arguments:EcTInt

Return Type:EcTInt

Privilege:Public

**WriteLogEntry** - Write a log entry to the log file

Arguments:

Return Type:EcTInt

Privilege:Public

### Associations:

The DpAtMgrLogData class has associations with the following classes:

Class: MgrGui

Class: DpAtMgrInstrConfigData GetLogFileLogical

Class:

DpAtMgr

PutActivityFlagGetActivityFlagWriteLogEntryReadLogEntryFindLogEntryGuiEditLog  
Annotation

Class: DpAtMgrCom ctor

#### 6.4.18 DpAtMgrProhibFuncListData Class

Parent Class:Not Applicable  
Public:No  
Distributed Object:No  
Purpose and Description:  
Stores data for prohibited function checker

##### Attributes:

**myLanguage** - Computer language of code to check: C, Fortran 77, Fortran 90 or Ada  
Data Type:EcTChar\*  
Privilege:Private  
Default Value:"\0"

**myNumProhibFuncs** - Number of prohibited functions for this language  
Data Type:EcTInt  
Privilege:Private  
Default Value:0

**myProhibFuncList** - Array of names of prohibited functions for a given language  
Data Type:EcTChar\*\*  
Privilege:Private  
Default Value:

##### Operations:

**\$DpAtProhibFuncListData** - Constructor  
Arguments:EcTChar\*  
Return Type:Void  
Privilege:Public

**GetNumProhibFuncs** - Gets number of prohibited functions from this class  
Arguments:EcTInt  
Return Type:Void  
Privilege:Public

**GetProhibFuncs** - Gets prohibited function list from this class storage  
Arguments:EcTChar\*\*

Return Type:Void  
Privilege:Public

**ReadProhibFuncs** - Reads prohibited function list from a file  
Arguments:EcTChar\*, EcTChar\*\*  
Return Type:Void  
Privilege:Private

**Associations:**

The DpAtMgrProhibFuncListData class has associations with the following classes:  
Class: DpAtMgrCheckProhibFuncCom

**6.4.19 DpAtPGERegistrationFile Class**

Parent Class:Not Applicable  
Public:No  
Distributed Object:No  
Purpose and Description:

This object represents a file that is input into the PgeRegistrationGui. It is made up of Pge Registration commands that can be parsed by the FileInput routine in DpAtPgeRegistrationGui.

**Attributes:**

None

**Operations:**

None

**Associations:**

The DpAtPGERegistrationFile class has associations with the following classes:  
Class: DpAtPgeRegistration Input  
Class: DpAtPgeRegistrationGui Input

## 6.4.20 DpAtPgeActivationRuleB Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class creates the PGE Activation Rule GUI. It allows the user to update, create, or change the Activation Rule of the current PGE.

### Attributes:

**DataScheduled** - This attribute is used for Data Scheduled PGEs. It holds the information needed to describe a Data Scheduled PGE.

Data Type:PIDataScheduled

Privilege:Private

Default Value:

**OrbitModel** - This attribute holds the definition of the Orbit Model for an Orbit Scheduled PGE.

Data Type:PIOrbitModel

Privilege:Private

Default Value:

**OrbitScheduled** - This attribute is used for an Orbit Scheduled PGE. It holds all the information needed to define an Orbit Scheduled PGE.

Data Type:PIOrbitScheduled

Privilege:Private

Default Value:

**PGEActivationType** - This is the Activation Type of the PGE. It has values of DataScheduled, TimeScheduled, OrbitScheduled, TileScheduled, and OtherScheduled.

Data Type:enum

Privilege:Private

Default Value:

**PGEProfileID** - This attribute holds the ID of the PGE in the PDPS database. It is used to access and update the PGE activation rule information.

Data Type:EctInt

Privilege:Private

Default Value:

**PIClusterList** - This attribute holds the Cluster and tile definitions for a Tile Scheduled PGE.

Data Type:List of PIClusters

Privilege:Private

Default Value:

**TileScheduled** - This attribute is used for Tile Scheduled PGEs. It holds all information required to define a Tile Scheduled PGE.

Data Type:PITileScheduled

Privilege:Private

Default Value:

**TimeScheduled** - This attribute is used for Time Scheduled PGEs. It holds the information needed to define a Time Scheduled PGE.

Data Type:PITimeScheduled

Privilege:Private

Default Value:

### Operations:

**DisplayActRuleAttr** - This method displays the attributes and their values from DataScheduled, TimeScheduled, OrbitScheduled, or TileScheduled PGEs based on the value in PGEActivationType.

Arguments:

Return Type:Void

Privilege:Private

PDL:

```
if (PGEActivationType = DataScheduled)
{
  // Loop through all attributes in the PIDataScheduled and display them and their
  // corresponding value to the screen.
}

if (PGEActivationType = TimeScheduled)
{
  // Loop through all attributes in the PIMTimeScheduled and display them and their
  // corresponding value to the screen.
}

if (PGEActivationType = OrbitScheduled)
{
  // Loop through all attributes in the PIOrbitScheduled and display them and their
  // corresponding value to the screen.
}

if (PGEActivationType = TileScheduled)
```

```

{
  // Loop through all attributes in the PITileScheduled and display them and their
  // corresponding value to the screen.
}

```

**DisplayOrbitModel** - This routine displays the Orbit Model for the PGE to the screen. Note that it does nothing if PGEActivationType is NOT OrbitScheduled.

Arguments:

Return Type:Void

Privilege:Public

PDL:

```

  // Loop through all attributes in the POrbitModel and display them and their
  // corresponding value to the screen.

```

**DisplayTiles** - This method displays the Clusters and their underlining tiles for the current PGE. Note that it does nothing if PGEActivationType is NOT TileScheduled.

Arguments:

Return Type:Void

Privilege:Public

PDL:

```

  // Loop through all attributes in the list of PIClusters and display them and their
  // corresponding value to the screen.

```

```

  // Loop through all PITiles associated with the cluster and display their
  // attributes and values.

```

**DpAtPgeActivationRule** - This is the constructor for the class. It creates the PGE Activation GUI. It takes in the PGEProfileID and calls ReadFromDatabase to retrieve the Activation Rule information for the PGE. It then calls DisplayActRuleAttr to list to the screen the current activation rules settings for the PGE. If the PGE is new, ReadFromDatabase is not called.

Arguments:EctInt PGEProfileID

Return Type:Void

Privilege:Public

PDL:

```

  // Create the Pge Activation Rule GUI via Builder Xccessory
  // generated code.

```

```

if (PGEProfileID != 0)

```

```

{

```

```

  // Call ReadFromDatabase to retrived the activation rule information

```

```

// for the PGE. Create the appropriate type of class based on the
// Activation Type of the PGE (DataScheduled, TimeScheduled, OrbitScheduled, or
// TileScheduled), and fill in the values with those in the database.

// Call DisplayActRuleAttr to display the attributes of the Activation Rule
// to the screen.

if (PGEActivationType = OrbitScheduled)
    // Call DisplayOrbitModel.
if (PGEActivationType = TileScheduled)
    // Call DisplayTiles.
}

```

**DpAtPgeActivationRule** - This constructor takes a Filename and Position in that file and calls FileInput to parse the file and make any Activation Rule changes specified in the file. It does not generate the Activation Rule GUI.

Arguments: EctInt PGEProfileID, RWCString Filename, EctInt Position

Return Type: Ectint

Privilege: Public

PDL:

```

if (PGEProfileID != 0)
{
    // Call ReadFromDatabase to retrieve the activation rule information
    // for the PGE. Create the appropriate type of class based on the
    // Activation Type of the PGE (DataScheduled, TimeScheduled, OrbitScheduled, or
    // TileScheduled), and fill in the values with those in the database.
}

// Call FileInput to process the input file and make any changes specified in
// the file.
FileInput (Filename, Position);

```

**FileInput** - This routine parses the file specified by Filename, from Position, and calls the other routines within this class to update the PGE's Activation Rule information based on the inputs from the file.

Arguments: RWCString Filename, EctInt Position

Return Type: Void

Privilege: Private

PDL:

```

while (not end of file)
{
    // Parse the file specified by Filename starting at Position.

    // Based on the type of command in the file, call the routine in this class that will

```

```

// perform the appropriate action, specifying Display=0.

if (Return from one of the routines != 0)
    // Exit loop.

if (Command in the file is not Activation Rule command)
    // Exit loop.
}

// Set return code to Position or "bad" value from called routine.

```

**ReadFromDatabase** - This routine reads the Activation Rule information from the PGE specified by PGEProfileID and places it in the appropriate local attribute(s).

Arguments:

Return Type:Void

Privilege:Private

PDL:

```

// Query the database for the PGE defined by the attribute PGEProfileID.

```

**SetPGEActivationType** - This method sets the PGE Activation Type. If Display is true, DisplayActRuleAttr is called to display a list of the attributes for that activation rule type.  
Arguments:enum PGEActivationType

**SubmitToDatabase** - This routine submits the PGE Activation rule information back to the database. It knows what attributes to store based on PGEActivationType.

Arguments:

Return Type:Void

Privilege:Public

PDL:

```

// Write the created/changed Activation class back to the PDPS database.

```

**UpdateActRuleAttr** - This routine updates the specified attribute with the specified value, based on the PGEActivationType. If Display is true, DisplayActRulAttr is called to list the change to the screen.

Arguments:RWCString Attribute, RWCString Value, EctInt Display

Return Type:Void

Privilege:Public

PDL:

```

// Call the Update routine of the appropriate class (based on the value
// of PGEActivationType) and pass it the specified Attribute and Value.

```

```
if (Display)
// Call DisplayActRuleAttr to display the changed attributes for the class.
DisplayActRuleAttr ();
```

**UpdateClusters** - This routine updates the attribute of the Cluster or underlining tiles with the specified value. If Display is true, DisplayTiles is called to redisplay the Cluster and Tiles to the screen with the change.

Arguments:RWCString Attribute, RWCString Value, EctInt Display

Return Type:Void

Privilege:Public

PDL:

```
// Call the Update routine of the PCluster class or the PTile class, and pass it the
specified
```

```
// Attribute and Value.
```

```
if (Display)
// Call DisplayTiles to display the changed attributes for the class.
DisplayTiles ();
```

**UpdateOrbitModel** - This routine updates the specified attribute to the specified value in the orbit model. If Display is true, DisplaOrbitModel is called to redisplay the orbit model to the screen.

Arguments:RWCString Attribute, RWCString Value, EctInt Display

Return Type:Void

Privilege:Public

PDL:

```
// Call the Update routine of the POrbitModel class and pass it the specified
// Attribute and Value.
```

```
if (Display)
// Call DisplayOrbitModel to display the changed attributes for the class.
DisplayOrbitModel ();
```

**~DpAtPgeActivationRule** - This the destructor for the class. It removes the PGE Activation Rule GUI from the display.

Arguments:

Return Type:Void

Privilege:Public

PDL:

```
// Remove the PGE Activation Rule GUI via Builder Xccessory
// generated code.
```

```
// Delete any Activation Classes created.
```

Associations:

The DpAtPgeActivationRuleB class has associations with the following classes:

Class: PIDataScheduled Create/Delete/Modify

Class: PIOrbitScheduledNB Create/Delete/Modify

Class: PITileScheduledNB Create/Delete/Modify

Class: PITimeScheduled Create/Delete/Modify

Class: DpAtPgeRegistrationGui Creates

Class: PICluster iscreated/deleted/modifiedby

Class: PIOrbitModelNB iscreated/deleted/modifiedby

Class: PITile iscreated/deleted/modifiedby

#### 6.4.21 DpAtPgeDataTypes Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class defines the PGE Data Type GUI that allows the user to add/delete/modify the data types used by a PGE for Input and Output.

##### Attributes:

**PGEInputs** - This is a list of the input data types of the PGE.

Data Type:List of PIDataTypeB

Privilege:Private

Default Value:

**PGEOutputs** - This is a list of the output data types of the PGE.

Data Type:List of PIDataTypeB

Privilege:Private

Default Value:

**PGEProfileID** - This is the database ID of the PGE used for retrieving and storing information from/to the PDPS database.

Data Type:EctInt

Privilege:Private

Default Value:

## Operations:

**CreateNewDataType** - This routine creates a new Data Type in the PDPS database. It takes in a Data Descriptor file specified by Filename, and then queries the user for any required information not provided in that file. If Display is true, the new data type is displayed to the screen along with the previously existing data types.

Arguments:RWCString Filename, EctInt Display

Return Type:Void

Privilege:Public

**DeleteInput** - This routine deletes the Input selected by the user (or specified by FileInput) from PGEInputs. If Display is true, then DisplayPgeInputs is called to display the change to the list.

Arguments:PIDataTypeB Input, EctInt Display

Return Type:Void

Privilege:Public

**DeleteOutput** - This routine deletes the Output selected by the user (or specified by FileInput) from PGEOutputs. If Display is true, DisplayPgeOutputs is called to show the change to the screen.

Arguments:PIDataTypeB Output, EctInt

Return Type:Void

Privilege:Public

**DisplayDataTypes** - This routine calls ReadFromDatabase to get a list of all of the Data Types in the PDPS database and displays them on the screen.

Arguments:

Return Type:Void

Privilege:Private

**DisplayPgeInputs** - This routine displays PGEInputs to the GUI.

Arguments:

Return Type:Void

Privilege:Private

**DisplayPgeOutputs** - This routine displays PGEOutputs to the GUI.

Arguments:

Return Type:Void

Privilege:Private

**DpAtPgeDataTypes** - This is the constructor for the class. It creates the PGE Data Type GUI. It also calls ReadFromDatabase to query the PDPS database for the PGEProfileID, and places the inputs and outputs of the PGE into PGEOutputs and PGEInputs.

Arguments:EctInt PGEProfileID

Return Type:Void  
Privilege:Public

**DpAtPgeDataTypes** - This constructor takes a Filename and Position in addition to the PGEprofileID, and calls FileInput to processing the specified file after querying the PDPS database for the PGE's inputs and outputs via a call to ReadFromDatabase.

Arguments:EctInt PGEprofileID, RWCString Filename, EctInt Position  
Return Type:EctInt  
Privilege:Public

**FileInput** - This routine takes the specified file and parses it from Position for any PGE Data Type commands. To perform the commands specified in the file it calls the other routines in this class.

Arguments:RWCString Filename, EctInt Position  
Return Type:EctInt  
Privilege:Private

**ReadFromDatabase** - This routine queries the PDPS database for the input and outputs of the PGE specified by PGEProfileID. It places the returned inputs and outputs in PGEInputs and PGEOutputs.

Arguments:  
Return Type:Void  
Privilege:Private

**SelectNewInput** - This routine takes the Input specified by the user (or in the input file being parsed by FileInput) and places it in PGEInputs. If Display is true, the new input is written to the screen.

Arguments:PIDataTypeB Input, EctInt Display  
Return Type:Void  
Privilege:Public

**SelectNewOutput** - This routine takes the Output selected by the user (or input from the file if called by FileInput) and places it in PGEOutputs. If Display is true, the new output is displayed to the screen via a call to DisplayPgeOutputs.

Arguments:PIDataTypeB Output, EctInt Display  
Return Type:Void  
Privilege:Public

**SubmitToDatabase** - This routine submits PGEInputs and PGEOutputs to the PDPS database.

Arguments:  
Return Type:Void  
Privilege:Public

**~DpAtPgeDataTypes** - This is the destructor for the class. It removes the PGE DataTypes

GUI from the screen.

Arguments:

Return Type:Void

Privilege:Public

**Associations:**

The DpAtPgeDataTypes class has associations with the following classes:

Class: PIAternate Create/Delete/Modify

Class: PIDataTypeReqB Create/Delete/Modify

Class: PIOutputYield Create/Delete/Modify

Class: PIDataTypeB Create/Modify

Class: DpAtPgeRegistrationGui Creates

### 6.4.22 DpAtPgeRegistration Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This represents the PGE Registration GUIs.

**Attributes:**

None

**Operations:**

None

**Associations:**

The DpAtPgeRegistration class has associations with the following classes:

Class: DpAtPGERegistrationFile Input

Class: MgrGui RunProgram

### 6.4.23 DpAtPgeRegistrationGui Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class defines the PGE Registration GUI that allows the user to populate the profile of the PGE in the PDPS database.

### Attributes:

**PGEDefinition** - This attribute contains the basic information needed to define a PGE to the PDPS database.

Data Type:PIPGE

Privilege:Private

Default Value:

**myPGEID** - This is the database identifier for the PGE. It is retrieved or assigned by the database.

Data Type:EctInt

Privilege:Private

Default Value:

**myPGEName** - This is the name of the PGE. It is input or selected by the user.

Data Type:RWCString

Privilege:Private

Default Value:

**myPGEVersion** - This is the version id of the PGE. It is selected or input by the user.

Data Type:EctFloat

Privilege:Private

Default Value:

**myPerformance** - This attribute defines the performance statistics of the PGE.

Data Type:PIPerformance

Privilege:Private

Default Value:

**myPerformanceFile** - This is a file that contains performance statistics produced during a run of the PGE.

Data Type:RWCString

Privilege:Private

Default Value:

**myResourceFile** - This is a file that contains resource usage of the PGE.

Data Type:RWCString  
Privilege:Private  
Default Value:

**myResourceUse** - This attribute defines the resource requirements for the PGE.

Data Type:PIResourceRequirement  
Privilege:Private  
Default Value:

## Operations:

**ChangePGEDefiniton** - This routine changes the attribute specified by Parameter to the specified Value in PGEDefinition. If Display is true, DisplayPGE is called to redisplay the information with the change.

Arguments:RWCString Parameter, RWCString Value, EctInt Display  
Return Type:Void  
Privilege:Public

**ChangePerformanceParameter** - This routine allows the user to change the specified performance parameter to the entered value.

Arguments:RWCString Parameter, RWCString Value, EctInt Display  
Return Type:Void  
Privilege:Public

**ChangeResourceParameter** - This routine allows the user to alter the specified resource parameter to the entered value.

Arguments:RWCString Parameter, RWCString Value, EctInt Display  
Return Type:Void  
Privilege:Public

**DisplayPGE** - This method displays the information in PGEDefintion to the screen.

Arguments:  
Return Type:Void  
Privilege:Public

**DpAtPgeRegistrationGui** - This is the constructor for the class. It creates the PGE Registration GUI.

Arguments:  
Return Type:EctVoid  
Privilege:Public

**DpAtPgeRegistrationGui** - This constructor takes in a file and calls FileInput to parse the file for PGE registration commands. The PGE Registration GUI is not created.

Arguments:RWCString Filename  
Return Type:EctInt  
Privilege:Public

**FileInput** - This routine parses the file specified by Filename and calls other routines in this class to process the PGE Registration commands.

Arguments:RWCString Filename  
Return Type:EctInt  
Privilege:Private

**ListPerformance** - This method displays the performance statistics of the current PGE defined in myPerformance. It displays the values to the GUI.

Arguments:  
Return Type:EctVoid  
Privilege:Public

**ListResource** - This method lists the resources currently defined for the PGE in myResourceUse. It displays those resources on the GUI.

Arguments:  
Return Type:EctVoid  
Privilege:Public

**ReadPerformanceFile** - This routine reads the performance file specified by Filename and stores them in myPerformance.

Arguments:RWCString Filename, EctInt Display  
Return Type:Void  
Privilege:Public

**ReadResourceFile** - This routine reads the resource file specified by Filename and stores the values in myResourceUse.

Arguments:RWCString Filename, EctInt Display  
Return Type:Void  
Privilege:Public

**ReadfromDatabase** - This routine queries the database for the PGE specified by PGENAME and Version (or PGEID), and returns the PGE Definition, Performance and Resource information.

Arguments:  
Return Type:EctInt  
Privilege:Private

**SetPGENAME** - This routine allows the user to set the name of the PGE. If the version of the PGE is also set, the PDPS database is queried for the user specified PGE, and if found, the PGE definition, Performance, and Resource information is retrieved. If the version has not been set, a list of versions for the specified name is returned to the screen.

Arguments:RWCString PGEName, EctInt Display  
Return Type:Void  
Privilege:Public

**SetPGEVersion** - This routine lets the user set the Version of the PGE. If the name of the PGE is also set, the PDPS database is queried for the PGE. If the PGE is found in the PDPS database, thePGE Definition, Performance, and Resource information is retrieved and displayed.

Arguments:RWCString PGEVersion, EctInt Display  
Return Type:Void  
Privilege:Public

**SubmittoDatabase** - This routine submits the PGE definition, performance, and resource information changes back to the PDPS database.

Arguments:  
Return Type:EctInt  
Privilege:Public

**UpdateActivationRules** - This method calls the constructor for the DpAtPgeActivationRulesB class to create the PGE Activation Rules GUI.

Arguments:RWCString Filename, EctInt Position  
Return Type:EctInt  
Privilege:Public

**UpdateDataTypes** - This method calls the constructor for the DpAtPgeDataTypes class to start the PGE Data Types GUI.

Arguments:RWCString Filename, EctInt Position  
Return Type:EctInt  
Privilege:Public

**UpdateUserParameters** - This method calls the constructor for the DpAtPgeUserParameters class to invoke the PGE User Parameters GUI.

Arguments:RWCString Filename, EctInt Position  
Return Type:Ectint  
Privilege:Public

**~DpAtPgeRegistrationGui** - This is the destructor for the class. It removes the PGE registration GUI from the screen.

Arguments:  
Return Type:Void  
Privilege:Public

## Associations:

The DpAtPgeRegistrationGui class has associations with the following classes:

- Class: PIPGE Creates/Deletes/Updates
- Class: PIPGEProfile Creates/Deletes/Updates
- Class: PIPPerformance Creates/Deletes/Updates
- Class: PIResourceRequirement Creates/Deletes/Updates
- Class: DpAtPgeActivationRuleB Creates
- Class: DpAtPgeDataTypes Creates
- Class: DpAtPGERegistrationFile Input
- Class: DpAtPgeUserParameters creates

### 6.4.24 DpAtPgeUserParameters Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class defines the PGE User Parameters GUI, that allows the user to add/delete/modify the user parameters for a PGE.

## Attributes:

**PgeProfileID** - This is the identifier of the PGE in the PDPS database.

Data Type:EctInt

Privilege:Private

Default Value:

**UserParameters** - This is the user parameters for the PGE.

Data Type:PIUserParameters

Privilege:Private

Default Value:

## Operations:

**AddNewParameter** - This routine adds a new User Parameter to UserParameters. If Display is true, then the new parameter is shown on the screen.

Arguments:RWCString Parameter, RWCString Value, EctInt Display

Return Type:EctInt

Privilege:Public

**DeleteParameter** - This routine deletes a parameter from UserParameters. If Display is true, the deleted parameter is removed from the GUI.

Arguments:RWCString Parameter, EctInt Display

Return Type:EctInt

Privilege:Public

**DisplayParameters** - This routine displays the parameters in UserParameters to the screen.

Arguments:

Return Type:Void

Privilege:Private

**DpAtPgeUserParameters** - This is the constructor for the class. It creates the PGE User Parameters GUI that allows the user to update the user parameters for a PGE. A call to ReadFromDatabase is made to retrieve the user Parameter information for the current PGE. If the user Parameter information exists in the database, DisplayParameters is called to list it to the screen.

Arguments:EctInt PgeProfileID

Return Type:EctVoid

Privilege:Public

**DpAtPgeUserParameters** - This constructor takes in a filename as well as a PgeId, and passes the file onto FileInput for parsing. This constructor does not create the PGE User Parameters GUI. A call to ReadFromDatabase is made with the argument PgeId to retrieve the user parameter information for the PGE.

Arguments:EctInt PgeProfileID, RWCString Filename, EctInt Position

Return Type:EctInt

Privilege:Public

**FileInput** - This routine parses the file specified by Filename from the Position in the file, and calls the other methods in this class to make the specified changes to the user parameters of the PGE.

Arguments:RWCString Filename, EctInt Position

Return Type:EctInt

Privilege:Private

**ModifyParameter** - This routine modifies the specified User Parameter with the specified value. If Display is true then the change is displayed to the screen.

Arguments:RWCString Parameter, RWCString Value, EctInt Display

Return Type:EctInt

Privilege:Public

**ReadFromDatabase** - This routine queries the PDPS database for the user parameters for the PGE specified by PGEProfileID and places them in UserParameters.

Arguments:

Return Type:EctInt  
Privilege:Private

**SearchForParameter** - This routine searches UserParameters for the specified parameter. If Display is true, and the parameter is found, it is highlighted on the GUI.

Arguments:RWCString Parameter, EctInt Display  
Return Type:Ectint  
Privilege:Public

**SubmitToDatabase** - This routine stores the attribute UserParameter back to the PDPS database.

Arguments:  
Return Type:EctInt  
Privilege:Public

**~DpAtPgeUserParameters** - This is the destructor for the class. It removes the PGe user Parameters GUI from the screen.

Arguments:  
Return Type:EctVoid  
Privilege:Public

#### **Associations:**

The DpAtPgeUserParameters class has associations with the following classes:

Class: PUserParameters Create/Delete/Update

Class: DpAtPgeRegistrationGui creates

#### **6.4.25 DpAtProcGui Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

GUI for starting a job in the Data Processing subsystem. **THIS IS NOT A CLASS.** It is callable from the Unix command line.

#### **Attributes:**

None

**Operations:**

None

**Associations:**

The DpAtProcGui class has associations with the following classes:

Class: MgrGui RunProgram

**6.4.26 DpAtSSAPFile Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This is a file with SSAP commands.

**Attributes:**

None

**Operations:**

None

**Associations:**

The DpAtSSAPFile class has associations with the following classes:

Class: DpAtSSAPGuiNB Input

**6.4.27 DpAtSSAPGuiNB Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class provides the primary access from AIT to the Science Software Archive Packages stored at the Data Server. It provides a GUI definition to allow the user to select an SSAP

to modify, create, or delete. It also allows the user the ability to activate the GUIs that allow for the modification of the files contained within an SSAP, or an SSAP's metadata.

### Attributes:

**myCurrentSSAP** - This attribute is the SSAP (from the list of SSAPs provided by the Data Server) selected by the user.

Data Type:EctInt

Privilege:Private

Default Value:

**mySSAP** - This is the currently selected and retrieved SSAP. If an SSAP is being created, then this is the SSAP under construction. Its type is defined by the SSAP type at the Data Server.

Data Type:DsNsSceinceSoftwareArchivePackage

Privilege:Private

Default Value:

**mySSAPFilters** - This attribute holds the filter information put into the GUI by the user. It is used to limit the listing of SSAPs to the screen. Its structure is that of an SSAP name (RWCString) and a version (EctFloat).

Data Type:structure

Privilege:Private

Default Value:

**mySSAPList** - This attribute holds the list of SSAPs retrieved from the Data Server and displayed on the GUI. Its structure is that of a list of SSAP names and versions.

Data Type:structure

Privilege:Private

Default Value:

### Operations:

**CreateSSAP** - This method creates a new SSAP. It prompts the user for the new SSAP name and version. If an existing SSAP has been chosen on the GUI, this routine prompts the user if he/she wants to copy the files from that SSAP to the new SSAP. The copy is performed if the user answers yes. Then the Edit SSAP File List GUI is started via a call to EditSSAPFileList.

Arguments:RWCString SSAPName, EctInt Display

Return Type:EctInt

Privilege:Public

PDL:

```
// Call CheckAccess to make sure the user is allowed to
```

```

// create an SSAP. If access is not allowed, then check Display
// parameter, and display a failure message.

if (SSAPName != null)
    // Call GetSSAP to retrieve the selected SSAP from the Data
    // Server.
    GetSSAP (SSAPName);

// Call RecordUpdate to log the fact that an SSAP has been created
// by this user.

if Display
{
    // Query the user for the name of the new SSAP.

    // Call the constructor for the EditSSAPFileListGui so that
    // the user can manipulate the files in the new SSAP.
    DpAtEditSSAPFileListGuiNB::DpAtEditSSAPFileListGuiNB (mySSAP);

    // Call ListSSAP to update the list of Science Software
    // Archive Packages on the display.
    ListSSAP();
}

```

**DeleteSSAP** - This method creates the Data Server command to delete the selected SSAP.

Arguments:RWCString SSAPName, EctInt Display

Return Type:EctInt

Privilege:Public

PDL:

```

// Call CheckAccess to make sure the user is allowed to
// delete an SSAP. If access is not allowed, then check Display
// parameter, and display a failure message.

// Create a Data Server command to delete the SSAP specified
// by SSAPName. Send the command to the Data Server.

// Call RecordUpdate to log the fact that an SSAP has been deleted
// by this user.

if Display
    // Call ListSSAP to update the list of Science Software
    // Archive Packages on the display.
    ListSSAP();

```

**DpAtSSAPGuiNB** - This is the constructor that creates the SSAP Gui. It will call CheckAccess from the DpAtAccess class to get the user's access level. The User's privileges will be displayed on the GUI. It also calls ListSSAPs to get a list of SSAPs for the user to work with.

Arguments:

Return Type: EctVoid

Privilege: Public

PDL: // Create SSAP GUI via Builder Xccessory generated  
// code.

// mySSAPFilters set to a null value.

// Initialize Data Server objects. Query Data Server  
// for list of Science Software Archive Packages (INSPECT).  
// Store the list of SSAPs in mySSAPList.

// Call ListSSAP to put the list of Science Software  
// Archive Packages on the display.  
ListSSAP();

**DpAtSSAPGuiNB** - This constructor allows the class to accept input from the Filename specified. It does not create the SSAP GUI, but parses the file for user actions relating to SSAPs. It initializes the interface with the Data Server and then calls other routines to parse the input file and make the specified modifications to an existing SSAP or to create a new one.

Arguments: RWCString Filename

Return Type: EctInt

Privilege: Public

PDL:

// mySSAPFilters set to a null value.

// Initialize Data Server objects. Query Data Server  
// for list of Science Software Archive Packages.  
// Store the list of SSAPs in mySSAPList.

// Call FileInput to parse the input file.  
FileInput (Filename);

**EditSSAPFiles** - This routine is called when the user chooses to Edit SSAP Files. It calls the constructor for DpAtEditSSAPFilesGuiNB.

Arguments:

```

Return Type:EctInt
Privilege:Public
PDL:  if (mySSAP = null)
    // Call GetSSAP to retrieve the selected SSAP from the Data
    // Server.
    GetSSAP (SSAPName);

    // Call the constructor for the EditSSAPFileListGui so that
    // the user can manipulate the files in the SSAP.
    DpAtEditSSAPFileListGuiNB::DpAtEditSSAPFileListGuiNB (mySSAP);

```

**EditSSAPFiles** - This method creates the DpAtEditSSAPFileListGuiNB class. It passes the Filename and current Position in that file onto the class so that file list changes can be made to the current SSAP.

Arguments:RWCString Filename, EctInt Position

Return Type:EctInt

Privilege:Public

PDL:

```

if (mySSAP = null)
    // Call GetSSAP to retrieve the selected SSAP from the Data
    // Server.
    GetSSAP (SSAPName);

    // Call the constructor for the EditSSAPFileListGui so that
    // any File List changes can be made via the input file.
    DpAtEditSSAPFileListGuiNB::DpAtEditSSAPFileListGuiNB (
        mySSAP, Filename, Position);

```

**EditSSAPMetadata** - This routine is called when the user selects Edit SSAP Metadata from the GUI. It calls the constructor for the DpAtEditSSAPMetadataGuiNB class.

Arguments:

Return Type:EctInt

Privilege:Public

PDL: if (mySSAP = null)

```

    // Call GetSSAP to retrieve the selected SSAP from the Data
    // Server.
    GetSSAP (SSAPName);

```

```

    // Call the constructor for the EditSSAPMetaGui so that
    // the user can manipulate the metadata of the SSAP.
    DpAtEditSSAPMetaGuiNB::DpAtEditSSAPMetaGuiNB ();

```

**EditSSAPMetadata** - This method creates the DpAtEditSSAPMetadataGuiNB class. It passes the Filename and current Position in that file onto the class so that metadata changes can be made to the current SSAP.

Arguments:RWCString Filename, EctInt Position

Return Type:EctInt

Privilege:Public

PDL:

```
if (mySSAP = null)
  // Call GetSSAP to retrieve the selected SSAP from the Data
  // Server.
  GetSSAP (SSAPName);

// Call the constructor for the EditSSAPMetaDataGui so that
// any File List changes can be made via the input file.
DpAtEditSSAPMetaDataGuiNB::DpAtEditSSAPMetaDataGuiNB (
  mySSAP, Filename, Position);
```

**FileInput** - This method parses the file specified and calls the routines of this class to make the SSAP changes specified.

Arguments:RWCString Filename

Return Type:EctInt

Privilege:Private

PDL:

```
// Open the file specified by Filename.

while (there is more to read in the file)
{
  // Parse each line in the file.
  switch
  {
    // For each type of file input command, call the
    // appropriate routine within this class to
    // process it.

    if (bad input from file)
      // Exit loop.

    if (bad return code from called routine)
      // Exit loop.
  }
}

// Close the file.
```

```
// return status.
```

**GetSSAP** - This method creates the Data Server command to Acquire the selected SSAP from the Data Server. It places it in mySSAP.

Arguments:RWCString SSAPName

Return Type:EctVoid

Privilege:Private

PDL:

```
// Create a Data Server command to ACQUIRE the SSAP specified
```

```
// by SSAPName. Send the command to the Data Server.
```

```
// Set mySSAP equal to the retrieved SSAP.
```

**ListSSAPs** - This routine is called when the GUI needs to display a list of SSAPs for the user to choose from. It makes a Data Server Query to get a list of all SSAPs that meets the filters specified in mySSAPFilters.

Arguments:

Return Type:EctVoid

Privilege:Private

PDL:

```
while (not done with the list)
```

```
{
```

```
  if mySSAPFilters != null
```

```
    // Check for the current entry in mySSAPList to match the
```

```
    // value in mySSAPFilters.
```

```
    if (match)
```

```
      // Display the name of the SSAP in position i of
```

```
      // mySSAPList.
```

```
    // Display the name of the SSAP in position i of
```

```
    // mySSAPList.
```

```
}
```

**SetSSAPFilters** - This method updates the SSAP filters for the display. It then calls the routine that displays the list of SSAPs, so that only those SSAPs matching the filter criteria appear on the GUI.

Arguments:structure GUIFilters

Return Type:EctVoid

Privilege:Public

PDL:

```

// Read filter information (SSAP name & version) from the display using
// Builder Xcessory generated code. Place into mySSAPFilters.
mySSAPFilters=GUIFilters;

// Call ListSSAPs to update the display.
ListSSAPs ();

```

**SubmitSSAP** - This routine creates the Data Server command to Insert mySSAP into the Data Server.

Arguments:RWCString SSAPName, EctInt Display

Return Type:EctInt

Privilege:Public

PDL:

```

// Create a Data Server command to INSERT the SSAP specified
// by SSAPName. Send the command to the Data Server.

```

```

// Set mySSAP to null because we are done with it.
mySSAP=null;

```

```

if Display

```

```

// Call ListSSAP to update the list of Science Software
// Archive Packages on the display.
ListSSAP();

```

**~DpAtSSAPGuiNB** - This is the destructor for the DpAtSSAPGuiNB class. It is called when the user exits the GUI.

Arguments:

Return Type:EctVoid

Privilege:Public

PDL:

```

// Call destructors for the other SSAP GUIs if they exist.
DpAtEditSSAPMetaDataGuiNB::~~DpAtEditSSAPMetaDataGuiNB ();
DpAtEditSSAPFileListGuiNB::~~DpAtEditSSAPFileListGuiNB ();

```

```

// Use Builder Xcessory generated code to remove the
// GUI from the screen.

```

```

// Delete Data Server objects and close the connection with
// the Data Server.

```

Associations:

The DpAtSSAPGuiNB class has associations with the following classes:

Class: GIParameterList Create/Insert

Class: DsCIESDTRReferenceCollector Create/SetCallback

Class: DsCICommand Create/SetCategory/SetServiceName

Class: GIParameter Create/SetParameter

Class: DsCIRequest Create/Submit

Class: DpAtEditSSAPFileListGuiNB Executes

Class: DpAtEditSSAPMetaDataGuiNB Executes

Class: DpAtSSAPFile Input

Class: DpAtAccessNB Query

Class: MgrGui executes

#### 6.4.28 DpPrAITManualIF Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class represents the Algorithm Integration & Test interface for the manual staging and destaging of data, algorithms, executables, or other stored items which have Universal References.

#### Attributes:

**myManualIFWindow** - This represents the GUI which will allow the AI&T position to manually stage and destage data products, executables, algorithms, etc.

Data Type:

Privilege:Private

Default Value:

#### Operations:

**Destage** - This operation is used by the Algorithm Integration & Test position to destage a piece of data, a PGE, an algorithm, or anything else which can be located by a UR.

Arguments:Item:GIUR

Return Type:Void

Privilege:Public

**Stage** - This operation stages the item located by the input UR. This can be a data granule,

a PGE, an algorithm, or anything else located by a UR.

Arguments:Item:GIUR

Return Type:Void

Privilege:Public

**Associations:**

The DpPrAITManualIF class has associations with the following classes:

Class: DsCIRequest Builds

Class: DsCIESDTRreferenceCollector SubmitsRequestThrough

### 6.4.29 DsCICommand Class

Parent Class:Not Applicable

**Attributes:**

None

**Operations:**

None

**Associations:**

The DsCICommand class has associations with the following classes:

Class: DpAtSSAPGuiNB Create/SetCategory/SetServiceName

DsCIRequest (Aggregation)

### 6.4.30 DsCIESDTRreferenceCollector Class

Parent Class:Not Applicable

Public:No

Distributed Object:Yes

Purpose and Description:

This public, distributed class is a specialization of the Collector class which handles DsCIESDTRreferences. This class is much more complex than the base class. This class

provides, in addition to the normal set operations for ESDTReferences, the ability to handle requests, working-collection synchronization, and sessions. It also contains private operations to hand the ESDTReference-level actions to the dataserver.

**Attributes:**

None

**Operations:**

None

**Associations:**

The DsCIESDTReferenceCollector class has associations with the following classes:

Class: DpAtSSAPGuiNB Create/SetCallback

Class: DpPrAITManualIF SubmitsRequestThrough

#### **6.4.31 DsCIRequest Class**

Parent Class:Not Applicable

**Attributes:**

None

**Operations:**

None

**Associations:**

The DsCIRequest class has associations with the following classes:

Class: DpPrAITManualIF Builds

Class: DpAtSSAPGuiNB Create/Submit

DsCIESDTReferenceCollector (Aggregation)

### 6.4.32 EosView Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

EosView program; displays contents of ECS HDF files. THIS IS NOT A CLASS. It is callable from the Unix command line.

#### Attributes:

None

#### Operations:

None

#### Associations:

The EosView class has associations with the following classes:

Class: MgrGui SpawnProgram

### 6.4.33 FORTRAN77codechecker Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

FORTRAN 77 code checker FORCHECK. A FORTRAN 77 code checker is necessary because most compilers (and presumably delivered F77 code) will not adhere strictly to the F77 ANSI standard. THIS IS NOT A CLASS. It is callable from the Unix command line.

#### Attributes:

None

#### Operations:

None

**Associations:**

The FORTRAN77codechecker class has associations with the following classes:

Class: MgrGui SpawnProgram

**6.4.34 Generalvisualizationtool Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This is an Abstract Class used tp represent the General data visualization tool IDL. It is callable from the Unix command line.

**Attributes:**

None

**Operations:**

None

**Associations:**

The Generalvisualizationtool class has associations with the following classes:

Class: MgrGui SpawnProgram

**6.4.35 GIPparameter Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class defines a parameter that is set in the request to the Data Server. One or more of these make up a GIPparameterList.

**Attributes:**

None

**Operations:**

None

**Associations:**

The GIPParameter class has associations with the following classes:

Class: DpAtSSAPGuiNB Create/SetParameter

**6.4.36 GIPParameterList Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class defines a list of parameters in a request to the Daat Server.

**Attributes:**

None

**Operations:**

None

**Associations:**

The GIPParameterList class has associations with the following classes:

Class: DpAtSSAPGuiNB Create/Insert

**6.4.37 Instrument-specificscript Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

Scripts written by users specific to an instrument configuration. Use for code check-in, compiling, running, etc. THIS IS NOT A CLASS. It is callable from the Unix command line.

**Attributes:**

None

**Operations:**

None

**Associations:**

The Instrument-specificscript class has associations with the following classes:

Class: MgrGui RunProgram

### 6.4.38 MgrGui Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

MgrGui is the main AIT Manager Gui. This class is necessarily ill-defined since it will contain code generated by the GUI builder.

**Attributes:**

**myMotifRcFileData** - Data read from Motif resources file, including menu labels, and program names and arguments where appropriate

Data Type:EcTChar\*\*

Privilege:Private

Default Value:

## Operations:

**DisplayReturnValue** - Displays value returned by system call on GUI

Arguments:EcTChar\*,EcTInt

Return Type:EcTInt

Privilege:Public

**Processor** - Main AIT Manager GUI processor

Arguments:DpAtMgrGuiActivityData,DpAtMgrChecklistData,DpAtMgrLogData

Return Type:EcTInt

Privilege:Public

**RunProgram** - Run a program which is callable from the Unix command line InputUnix program name and options as a single string \*\*\*\*\*PDL\*\*\*\*\* Make system call to run program Wait for program to execute Return same return value from program

Arguments:EcTChar\*

Return Type:EcTInt

Privilege:Public

**SpawnProgram** - Spawns a Unix program or script. Immediately returns a return value without waiting for program to execute.

Arguments:EcTChar\*

Return Type:EcTInt

Privilege:Public

## Associations:

The MgrGui class has associations with the following classes:

Class: DpAtMgrLogData

Class: DpAtMgrCom Processorctor()

Class: DpAtMgrChecklistData PutActivityFlagPutCurrentIndex

Class: DpAtMgrGuiActivityData

PutSelectedAreaPutMenuSelectionGetActivityRequestPutProgramReturnValue

Class: CMscript RunProgram

Class: DpAtMgrCheckHdfFile RunProgram

Class: DpAtPgeRegistration RunProgram

Class: DpAtProcGui RunProgram

Class: Instrument-specificscript RunProgram

Class: Analysisenvironment SpawnProgram

Class: EosView SpawnProgram

Class: FORTRAN77codechecker SpawnProgram

Class: Generalvisualizationtool SpawnProgram

Class: Postscriptfileviewer SpawnProgram

Class: Text-graphicsviewer SpawnProgram  
Class: Webbrowser SpawnProgram  
Class: Windowsemulator SpawnProgram  
Class: xterm SpawnProgram  
Class: DpAtMgrBinaryFileEnvironmentGui executes  
Class: DpAtSSAPGuiNB executes

#### **6.4.39 PIAAlternate Class**

Parent Class:Not Applicable  
Public:No  
Distributed Object:No  
Purpose and Description:  
This object defines alternate inputs for a PGE. It is a PLS class.

##### **Attributes:**

None

##### **Operations:**

None

##### **Associations:**

The PIAAlternate class has associations with the following classes:  
Class: DpAtPgeDataTypes Create/Delete/Modify

#### **6.4.40 PICluster Class**

Parent Class:Not Applicable  
Public:No  
Distributed Object:No  
Purpose and Description:  
This class defines a group of Tiles. It is a PLS class.

##### **Attributes:**

None

**Operations:**

None

**Associations:**

The PICluster class has associations with the following classes:

Class: DpAtPgeActivationRuleB iscreated/deleted/modifiedby

#### **6.4.41 PIDataScheduled Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class defines a Data Scheduled PGE. It is a PLS class.

**Attributes:**

None

**Operations:**

None

**Associations:**

The PIDataScheduled class has associations with the following classes:

Class: DpAtPgeActivationRuleB Create/Delete/Modify

#### **6.4.42 PIDataTypeB Class**

Parent Class:Not Applicable

Public:Yes

Distributed Object:No

Persistent Class:True

**Purpose and Description:**

This class describes a data type known to the planning subsystem. This is a description of an input or output type, distinct to a granule or instance of the data type. The class is an abstraction or proxy that describes one of the Data Server ESDTs. The class captures data and operations that are required to subscribe and receive notification from the Data Server when a new instance of the Data Type arrives.

**Attributes:**

None

**Operations:**

None

**Associations:**

The PIDataTypeB class has associations with the following classes:

Class: DpAtPgeDataTypes Create/Modify

#### **6.4.43 PIDataTypeReqB Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

**Purpose and Description:**

This class defines the required inputs for a PGE. It is a PLS class.

**Attributes:**

None

**Operations:**

None

**Associations:**

The PIDataTypeReqB class has associations with the following classes:

Class: DpAtPgeDataTypes Create/Delete/Modify

#### 6.4.44 PLOrbitModelNB Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class defines an Orbit Model for an Orbit Scheduled PGE. It is a PLS class.

##### **Attributes:**

None

##### **Operations:**

None

Associations:

The PLOrbitModelNB class has associations with the following classes:

Class: DpAtPgeActivationRuleB iscreated/deleted/modifiedby

#### 6.4.45 PLOrbitScheduledNB Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class defines an Orbit Scheduled PGE. It is a PLS class.

##### **Attributes:**

None

##### **Operations:**

None

Associations:

The PLOrbitScheduledNB class has associations with the following classes:

Class: DpAtPgeActivationRuleB Create/Delete/Modify

#### 6.4.46 PIOutputYield Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class defines the expected output yield of a PGE's output. It is a PLS class.

##### **Attributes:**

None

##### **Operations:**

None

##### **Associations:**

The PIOutputYield class has associations with the following classes:

Class: DpAtPgeDataTypes Create/Delete/Modify

#### 6.4.47 PIPGE Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This is the base class within a generalization hierachy that describes PGEs. The class defines abstract operations required for the palnning subsystem to work out when a GE needs to be scheduled as well as containing the key attributes defining the PGE.

##### **Attributes:**

None

##### **Operations:**

None

**Associations:**

The PIPGE class has associations with the following classes:

Class: DpAtPgeRegistrationGui Creates/Deletes/Updates

**6.4.48 PIPGEProfile Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class represents a specific PGE Profile. It is a PLS class.

**Attributes:**

None

**Operations:**

None

**Associations:**

The PIPGEProfile class has associations with the following classes:

Class: DpAtPgeRegistrationGui Creates/Deletes/Updates

**6.4.49 PIPPerformance Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class contains the basic information that defines a PGE to PDPS. It is a PLS class.

**Attributes:**

None

**Operations:**

None

**Associations:**

The PIPerformance class has associations with the following classes:

Class: DpAtPgeRegistrationGui Creates/Deletes/Updates

**6.4.50 PIResourceRequirement Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class defines the performance statistics for a PGE. It is a PLS class.

**Attributes:**

None

**Operations:**

None

**Associations:**

The PIResourceRequirement class has associations with the following classes:

Class: DpAtPgeRegistrationGui Creates/Deletes/Updates

**6.4.51 PITile Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class defines a Tile, a geographic location for a PGE to process. It is a PLS class.

**Attributes:**

None

**Operations:**

None

**Associations:**

The PITile class has associations with the following classes:

Class: DpAtPgeActivationRuleB iscreated/deleted/modifiedby

**6.4.52 PITileScheduledNB Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class defines a Tile Scheduled PGE. It is a PLS class.

**Attributes:**

None

**Operations:**

None

Associations:

The PITileScheduledNB class has associations with the following classes:

Class: DpAtPgeActivationRuleB Create/Delete/Modify

**6.4.53 PITimeScheduled Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class defines a Time Scheduled PGE. It is a PLS class.

**Attributes:**

None

**Operations:**

None

**Associations:**

The PTimeScheduled class has associations with the following classes:

Class: DpAtPgeActivationRuleB Create/Delete/Modify

#### **6.4.54 PIUserParameters Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This class defines the User Parameters for the PGE and their default values to the PDPS database. It is a PLS class.

**Attributes:**

None

**Operations:**

None

**Associations:**

The PIUserParameters class has associations with the following classes:

Class: DpAtPgeUserParameters Create/Delete/Update

### 6.4.55 Postscriptfileviewer Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This is an Abstract Class used to represent the PostScript file viewer Ghostview. THIS IS NOT A CLASS. It is callable from the Unix command line.

#### Attributes:

None

#### Operations:

None

#### Associations:

The Postscriptfileviewer class has associations with the following classes:

Class: MgrGui SpawnProgram

### 6.4.56 Text-graphicsviewer Class

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This is an abstract class used to represent the Text and graphics Adobe Acrobat. It is callable from the Unix command line.

#### Attributes:

None

#### Operations:

None

**Associations:**

The Text-graphicsviewer class has associations with the following classes:

Class: MgrGui SpawnProgram

**6.4.57 Webbrowser Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This is an abstract class used to represent WorldWideWeb browser Mosaic. It is callable from the Unix line.

**Attributes:**

None

**Operations:**

None

**Associations:**

The Webbrowser class has associations with the following classes:

Class: MgrGui SpawnProgram

**6.4.58 Windowsemulator Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

Run SoftWindows DOS/Windows emulator. THIS IS NOT A CLASS. It is callable from the Unix command line.

**Attributes:**

None

**Operations:**

None

**Associations:**

The Windowsemulator class has associations with the following classes:

Class: MgrGui SpawnProgram

**6.4.59 xterm Class**

Parent Class:Not Applicable

Public:No

Distributed Object:No

Purpose and Description:

This is an Abstract Class used to represent an Unix xterm. CM (ClearCase) view is set automatically since a view is already up when AIT Manager is invoked. It is callable from the Unix command line.

**Attributes:**

None

**Operations:**

None

**Associations:**

The xterm class has associations with the following classes:

Class: MgrGui SpawnProgram

## **6.5 CSCI Dynamic Model**

### **6.5.1 AIT Manager GUI Scenarios**

This section gives scenarios for use of the AIT Manager GUI. These scenarios cover various operations which will be invoked from the AITTL Manager GUI. These operations include initiating one of the many COTS tools which will be provided for integration and test activities.

#### **6.5.1.1 Display Main AIT Manager GUI**

##### **6.5.1.1.1 Abstract**

This scenario occurs whenever the AIT Manager is invoked. Information used to set up the AI&T Manager GUI is gathered from various input files. This information sets the various user preferences invoked for the GUI.

##### **6.5.1.1.2 Stimulus**

The initial stimulus is invoked by the Algorithm Integration and Test Operations staff who initiates the AIT Manager program name and options from the UNIX command line. Before this operation can occur, the Configuration Management (ClearCase) view must have been set previously.

##### **6.5.1.1.3 Desired Response**

Main AIT Manager GUI, including checklist and log, is displayed on user's screen.

##### **6.5.1.1.4 Participating Classes from the Object Model**

- DpAtMgrCom
- DpAtMgrCmdLineData
- DpAtMgrInstrConfigData
- DpAtMgrChecklistData
- DpAtMgrLogData
- DpAtMgr
- DpAtMgrGuiActivityData
- DpAtMgrGui

##### **6.5.1.1.5 Scenario Description**

- a. User logs into AIT development machine
- b. User sets a CM (ClearCase) view
- c. User types in AIT Manager executable name and options, if any
- d. The default command line options are read from a file; instrument name, instrument config file logical, and static Motif resources file logical.
- e. If user typed in any command line options, these options overwrite the defaults, and the new defaults are saved to a file.

- f. The instrument configuration data is read from a file: script names, options, menu labels, file logicals, checklist, and log file logicals.
- g. The Static Motif resources file is read (all AIT Manager menu data except RUN menu: menu names, locations, program names and options if applicable), and the dynamic Motif resources file is written. This file consists of static motif resources data and instrument config data for RUN menu item.
- h. Checklist file data and the last log entry is read.
- i. Main AIT Manager GUI is displayed, using dynamic Motif resources file, checklist data and last log entry data.

#### **6.5.1.1.5 Event Trace**

Figure 6.5-1 shows the display AI&T main GUI event trace.

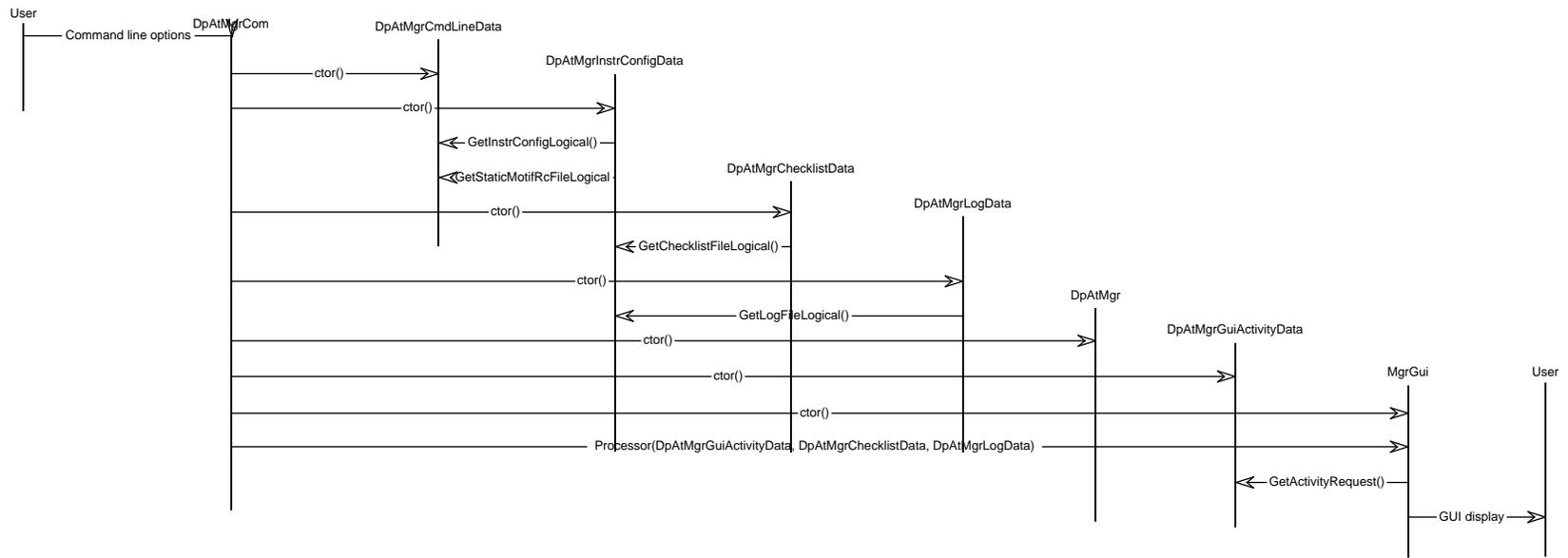


Figure 6.5-1. Display AIT Manager GUI Event Trace

## 6.5.1.2 Select a TOOLS Menu Item

### 6.5.1.2.1 Abstract

This scenario details what happens when the user selects TOOLS menu item *xterm*. The scenario applies equally to the other TOOLS menu items, which are COTS programs (unless noted) useful in the AIT environment:

- a. General visualization (IDL)
- b. Web browser (Mosaic)
- c. Text-graphics viewer (Acrobat)
- d. PostScript file viewer (Ghostview)
- e. Windows emulator (SoftWindows)
- f. Analysis environment (SPARCWorks on AIT Sun Workstation, CODEVision on AIT SGI Workstation)
- g. HDF file visualization (EosView, an ECS custom application)
- h. Fortran 77 code checker (FORCHECK)
- i. Dynamic memory leak detector

### 6.5.1.2.2 Stimulus

User pulls down TOOLS menu, clicks on *xterm*.

### 6.5.1.2.3 Desired Response

X-Windows system *xterm* is displayed. CM (ClearCase) view is automatically set by CM.

### 6.5.1.2.4 Participating Classes From the Object Model

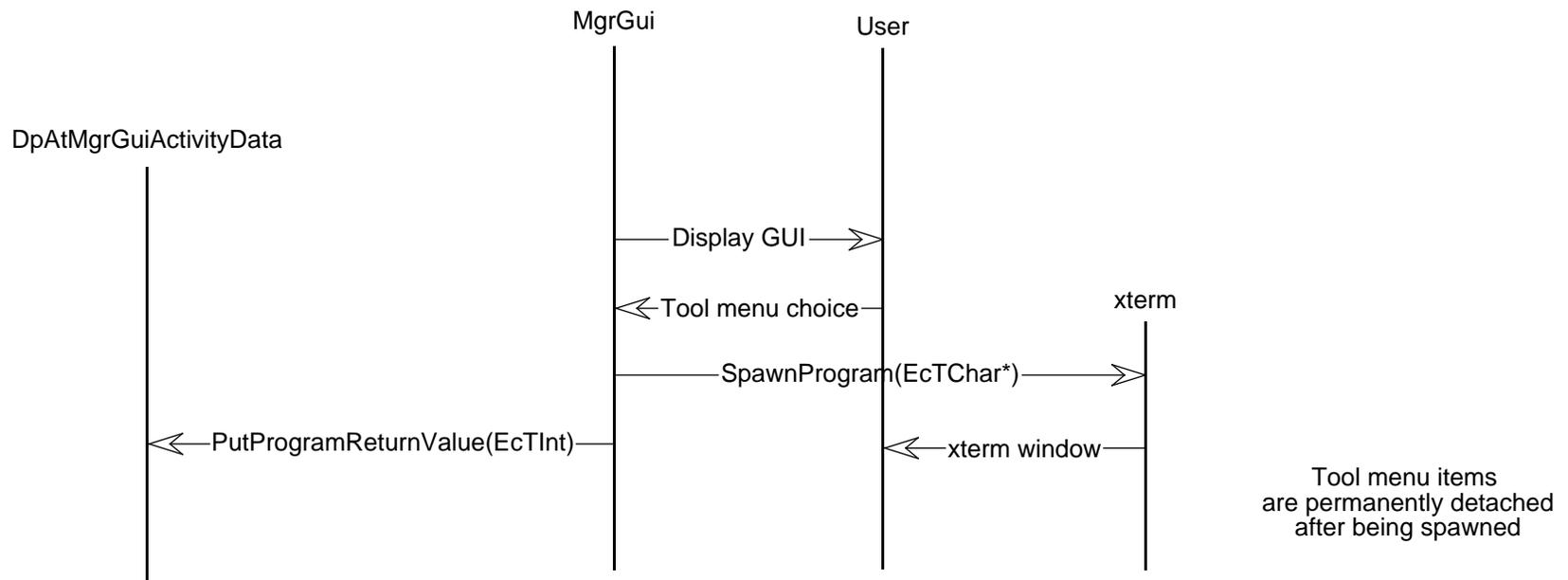
- DpAtMgrGui

### 6.5.1.2.5 Scenario Description

- a. AIT Manager screen previously displayed
- b. User pulls down TOOLS menu
- c. User clicks on *xterm*
- d. UNIX *xterm* is displayed, which automatically has a CM (ClearCase) view set
- e. status value is immediately returned indicating whether *xterm* was successfully spawned
- f. User types commands into *xterm*
- g. Other AIT Manager selections are always available during this time
- h. *xterm* remains up until user kills it manually or logs off the machine

### 6.5.1.2.6 Event Trace

Figure 6.5-2 shows the run tools menu event trace.



Tool menu items  
are permanently detached  
after being spawned

Example applies to  
all TOOLS menu items

**Figure 6.5-2. Run Tools Item Event Trace**

### 6.5.1.3 Select a UTILITIES Menu Item

#### 6.5.1.3.1 Abstract

This scenario represents activities that occur when the user selects UTILITIES menu item DpAtMgrCheckHdfFile. It applies equally to other UTILITIES menu items, which are all ECS custom programs:

- PGE Registration GUI (DpAtPgeRegistration)
- CM scripts
- DpAtMgrCheckPcfGui
- DpAtMgrBinaryFileEnvironmentGui
- DpAtMgrCheckProhibFuncGui
- DpAtMgrSSAPGuiNB

This scenario also applies to RUN menu items:

- Instrument-specific scripts

#### 6.5.1.3.2 Stimulus

The Algorithm Integration and Test Operations Staff wants to compare versions of HDF files generated at the SCF to the HDF files generated at the DAAC with SCF Toolkit linked in.

#### 6.5.1.3.3 Desired Response

HDF file checker GUI is displayed

#### 6.5.1.3.4 Participating Classes from the Object Model

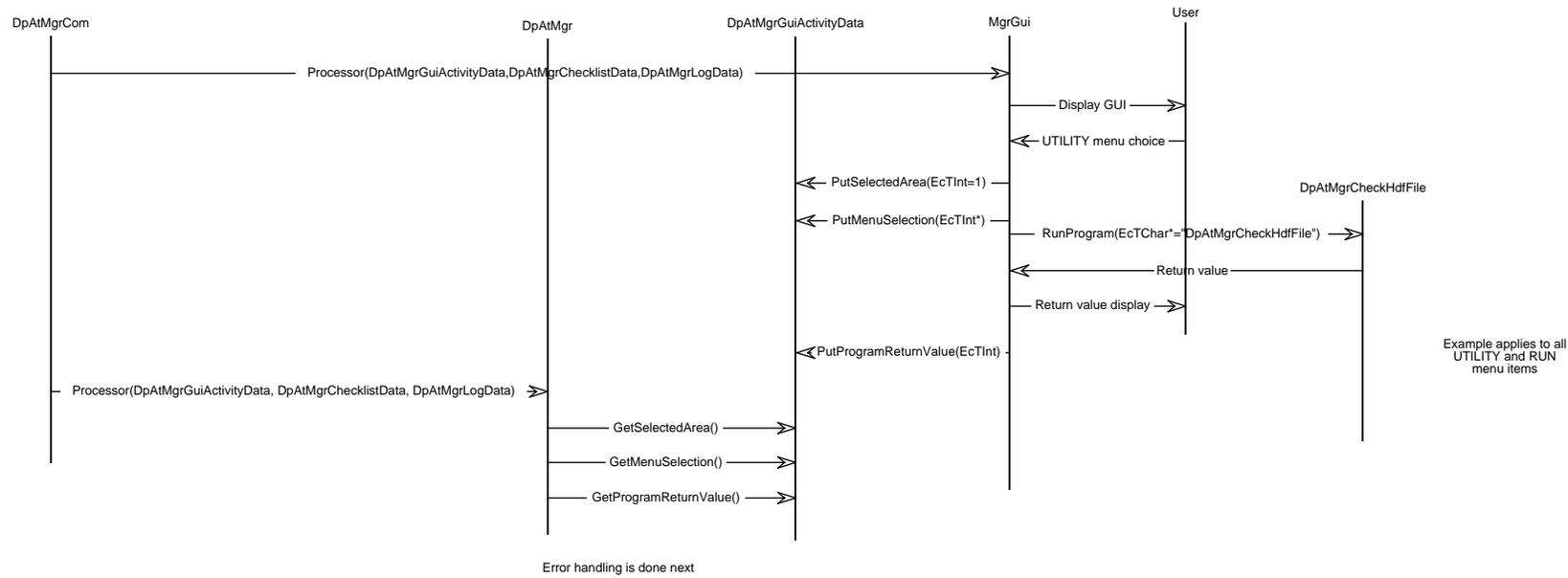
- DpAtMgrCom
- DpAtMgr
- DpAtMgrGuiActivityData
- DpAtMgrGui
- DpAtMgrCheckHdfFil

#### 6.5.1.3.5 Scenario Description

- a. AIT Manager screen previously displayed
- b. User pulls down UTILITIES menu
- c. User clicks on *HDF file checker*
- d. HDF file checker GUI is displayed
- e. User makes GUI selections, to compare HDF files and/or display HDF metadata
- f. Other AIT Manager selections are unavailable during this time
- g. User quits from HDF file checker GUI
- h. Return value from GUI triggers error handling, if necessary
- i. AIT Manager waits for another user selection

#### 6.5.1.3.6 Event Trace

Figure 6.5-3 shows the run utility menu item event trace.



**Figure 6.5-3. Run Utility Menu Item Event Trace**

#### **6.5.1.4 Select a Checklist Item**

##### **6.5.1.4.1 Abstract**

The AIT Manager checklist is a display of items required for check-in, compilation, test execution, etc. of science software. It is displayed as a line of text for each item, with a check box that is checked or unchecked. Each item has a unique ID; the checklist is specific to an instrument configuration. The checklist is manual, in that the user must check it with a mouse click; the program does not check any boxes automatically. Each time a box is checked, the log is updated.

This scenario shows what happens when a box for an item is checked.

##### **6.5.1.4.2 Stimulus**

User wants to record that compile and link to SCF Toolkit was successful.

##### **6.5.1.4.3 Desired Response**

Box is displayed as checked; log is updated. Item is noted as checked in checklist file after AIT Manager exits.

##### **6.5.1.4.4 Participating Classes from the Object Model**

- DpAtMgrCom
- DpAtMgr
- DpAtMgrGuiActivityData
- DpAtMgrGui
- DpAtMgrChecklistData
- DpAtMgrLogData

##### **6.5.1.4.5 Scenario Description**

- a. AIT Manager has been previously displayed
- b. User separately compiles and links science software with SCF Toolkit, e.g., by instrument-specific script or CM script
- c. User clicks on check box for "Compile and link with SCF Toolkit" item
- d. Checklist item state is changed from unchecked to checked internally
- e. Checked box is displayed
- f. New log entry is created, displayed on screen, and written to log file

##### **6.5.1.4.6 Event Trace**

Figure 6.5-4. shows the select checklist item event trace.

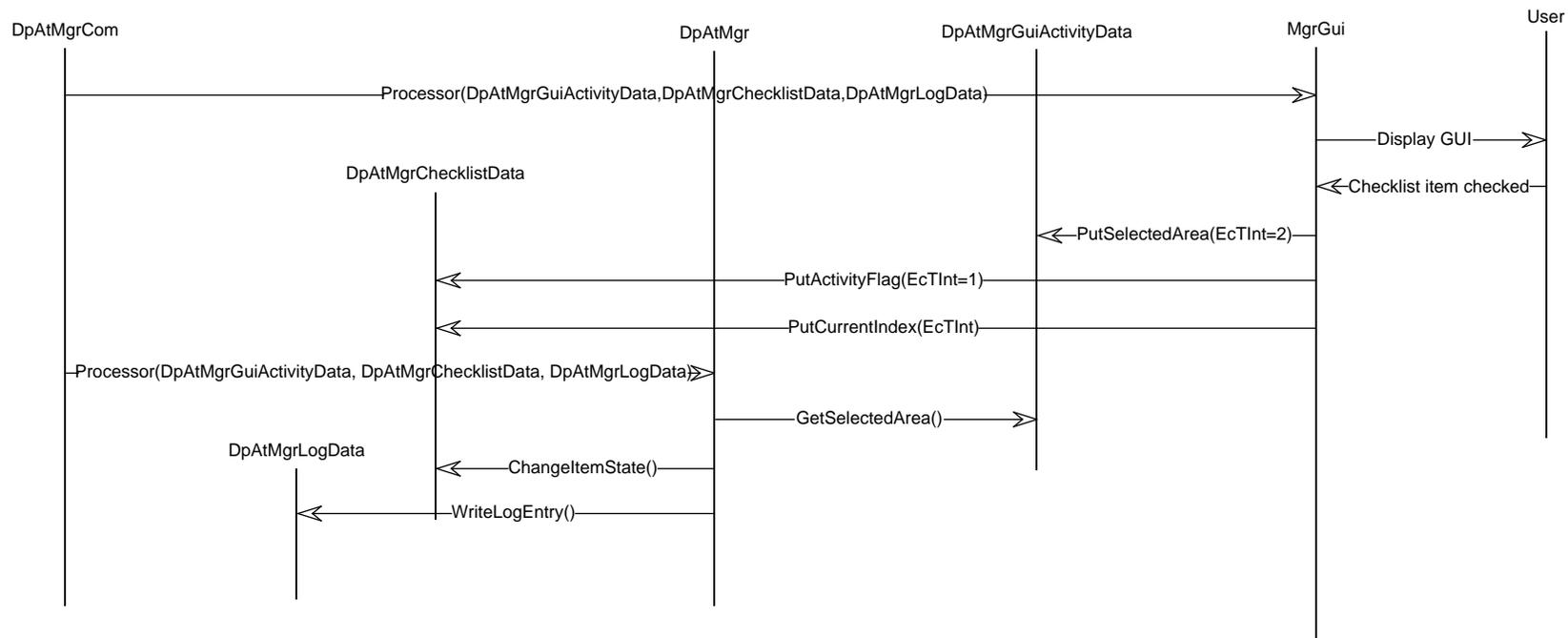


Figure 6.5-4. Select Checklist Item

## **6.5.2 Acquiring and Inserting Data Scenarios**

This section describes the scenario for the AITTL ability to allow the user to Acquire data from or Insert data to the Data Server.

### **6.5.2.1 Submit Staging or Destaging Request**

#### **6.5.2.1.1 Abstract**

These scenarios describe the functions provided to manually initiate the staging (Acquire) or destaging (Insert) of data, i.e., Science software, Test Data, etc., which are archived in the Data Server archive. The steps taken are very similar to activities performed in the Processing CSCI to automatically initiate staging or destaging by a software application. This will be added to the Ir-1 provided functionality for Release B.

#### **6.5.2.1.2 Stimulus**

DAAC Operations Staff initiates the staging or destaging of data.

#### **6.5.2.1.3 Desired Response**

Data is staged or destaged.

#### **6.5.2.1.4 Participating Classes from the Object Model**

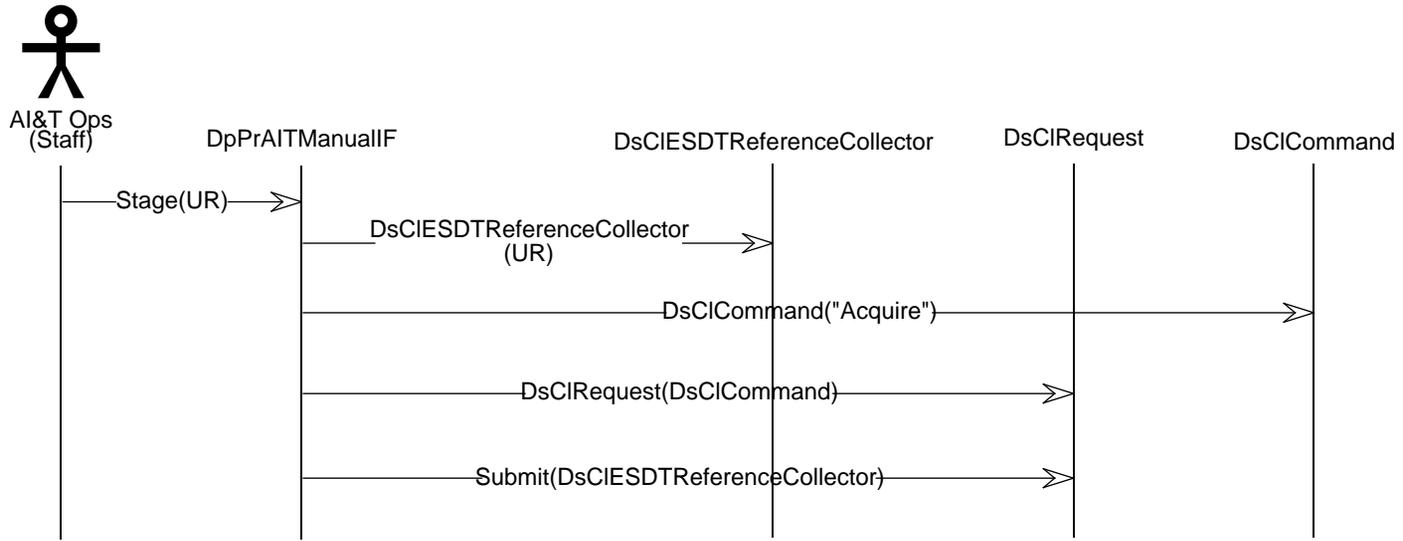
- DpPrAITManualIF
- DsCIESDTRreferenceCollector
- DsCIRequest
- DSCICommand

#### **6.5.2.1.5 Scenario Description**

- a. AIT Manager has been previously displayed
- b. User Requests the staging or destaging of a given data item, using specified reference materials as input guidance.
- c. Data Server copies data to specified storage location.
- d. Data Server informs AITTL when completed.

#### **6.5.2.1.6 Event Trace**

Figure 6.5-5 shows the submit staging request event trace. Figure 6.5-6 shows the submit destaging request event trace.



**Figure 6.5-5. Submit Staging Request**

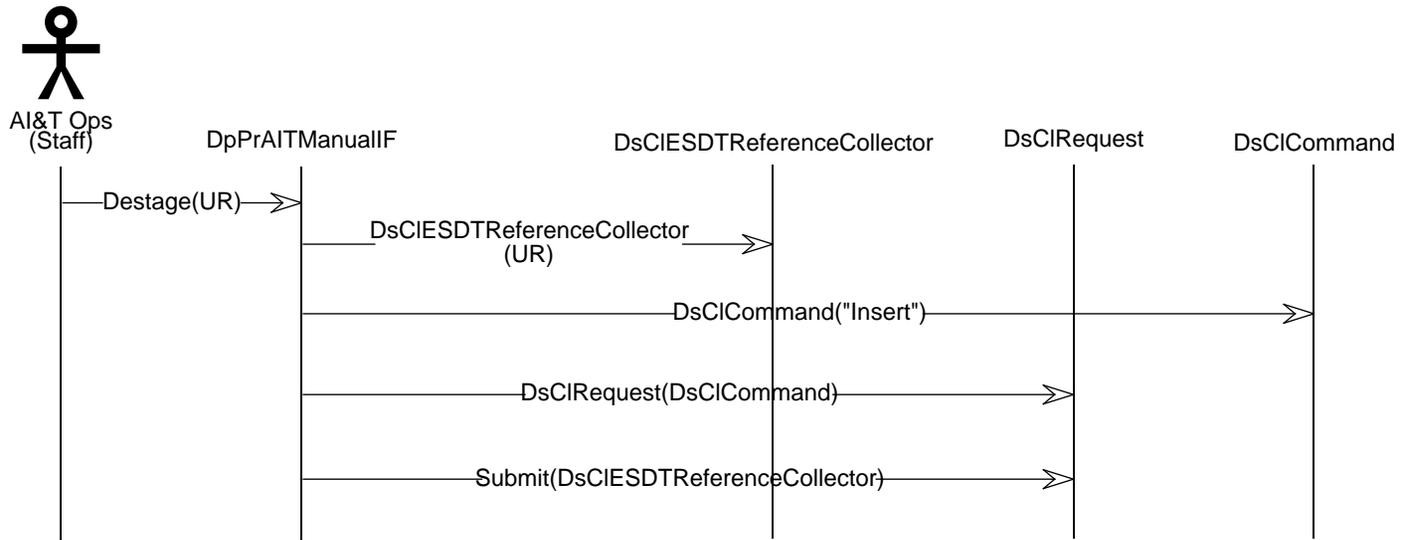


Figure 6.5-6. Submit Destaging Request

### **6.5.3 Science Software Archive Package GUI Scenarios**

This section describes scenarios for the uses of the AITTL Science Software Archive Package GUIs.

#### **6.5.3.1 Starting the AITTL Science Software Archive Package (SSAP) GUI**

##### **6.5.3.1.1 Abstract**

This scenario describes the execution of the SSAP GUI from a choice on the AIT Manager GUI.

##### **6.5.3.1.2 Stimulus**

DAAC Operations Staff want to manipulate an SSAP at the Data Server, so they choose the SSAP GUI option from the UTILITY menu on the AIT Manager GUI.

##### **6.5.3.1.3 Desired Response**

The SSAP GUI is placed on the screen with a list of SSAPs stored at the Data Server.

##### **6.5.3.1.4 Participating Classes from the Object Model**

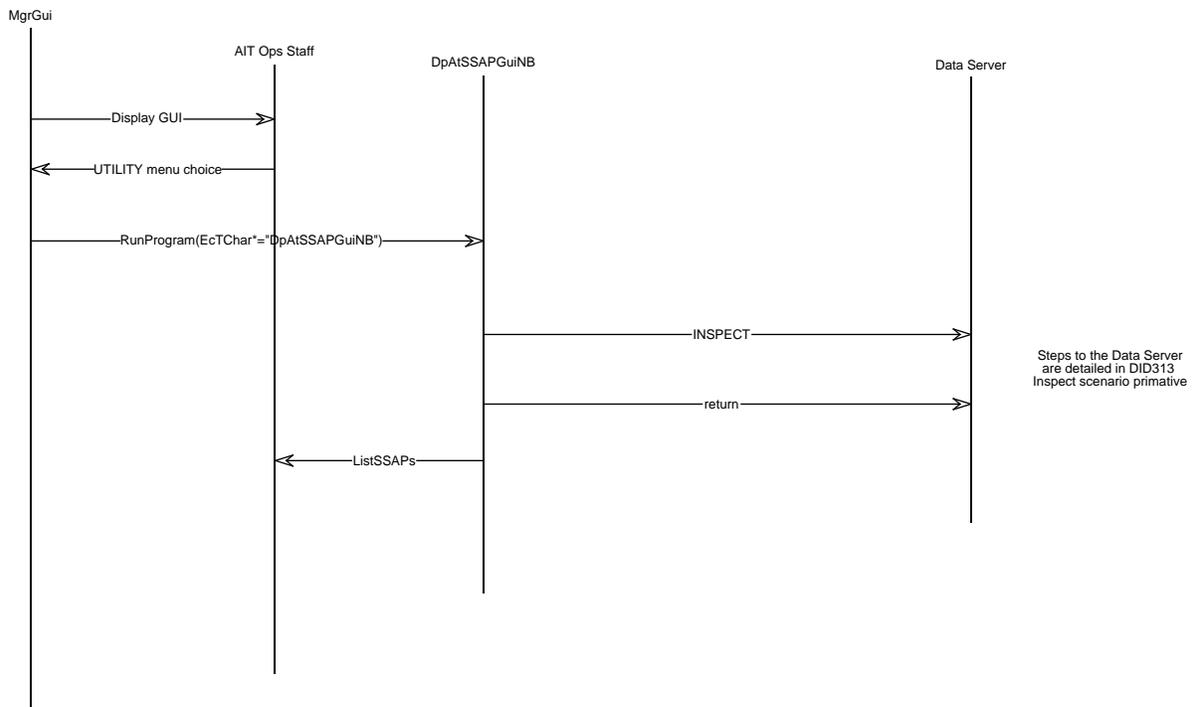
- DpAtMgrGui
- DpAtSSAPGuiNB

##### **6.5.3.1.5 Scenario Description**

- a. AIT GUI Manager has been previously displayed.
- b. User pulls down UTILITIES menu.
- c. User clicks on *Science Software Archive Package Tools*.
- d. Science Software Archive Package GUI is displayed. An Inspect command is made of the Data Server and a list of existing SSAPs is returned. The list of SSAPs is displayed to the screen.

##### **6.5.3.1.6 Event Trace**

Figure 6.5-7 shows the execution of the SSAP GUI from the AIT Manager.



**Figure 6.5-7. Create the Science Software Archive Package GUI**

## 6.5.3.2 Editing the List of Files within a Chosen Science Software Archive Package

### 6.5.3.2.1 Abstract

This scenario describes the manipulation of the files within an SSAP via the Edit SSAP File List GUI.

### 6.5.3.2.2 Stimulus

DAAC Operations Staff want to add and delete files from an SSAP. After starting the SSAP GUI (described in Section 6.5.3.1), the staff selects an SSAP from the displayed list, and chooses the menu choice to Edit the SSAP File List. Choices are made on the Edit SSAP File List GUI to add a file to, and delete a file from the chosen SSAP.

### 6.5.3.2.3 Desired Response

The chosen files are added to and deleted from the SSAP.

### 6.5.3.2.4 Participating Classes from the Object Model

- DpAtSSAPGuiNB
- DpAtEditSSAPFileListGuiNB
- DpAtAccessNB

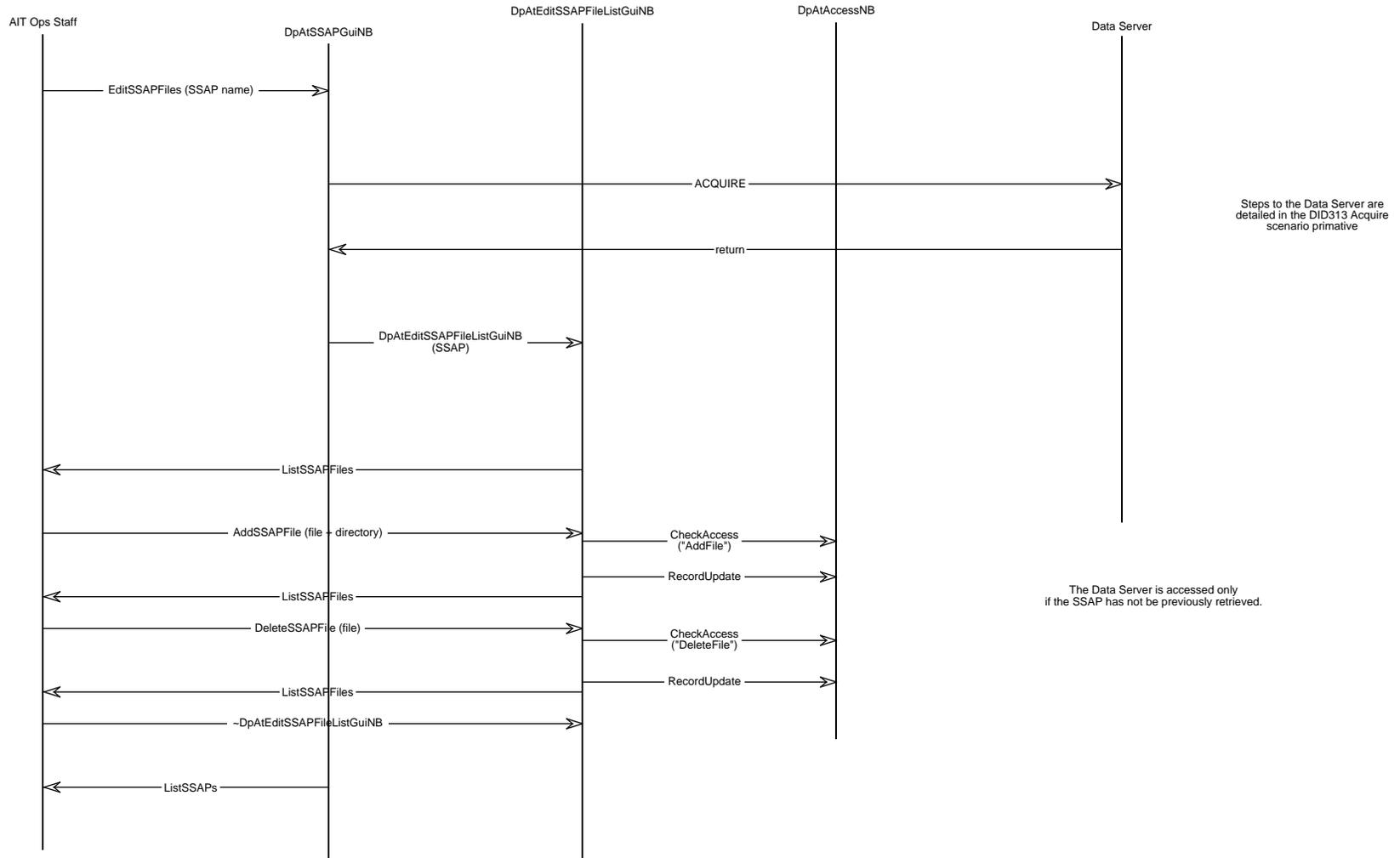
### 6.5.3.2.5 Scenario Description

- a. Science Software Archive Package GUI has been previously displayed, complete with a list of Science Software Archive Packages stored at the Data Server.
- b. User selects one of the SSAPs.
- c. User clicks on *Files*.
- d. An ACQUIRE command is formed and sent to the Data Server to retrieve the selected SSAP. The SSAP is placed on the local processor by the Data Server.
- e. Science Software Archive Package File List GUI is displayed. A list of all of the files contained in the selected and retrieved SSAP is placed on the screen.
- f. User updates the *Directory Location* for the files to Add/Replace and selects one of the files listed in the window. User then clicks *Add*.
- g. The access permissions of the User for the Add File capability are checked. It is assumed that the User has permission to add a file to the SSAP. The fact that the user is adding a file to the SSAP is recorded in the Activity Log. If the user does not have permission to add files to an SSAP an error message is displayed to the screen and step (h) is not done.
- h. The selected file is added to the SSAP. The list of files in the SSAP is updated to included the added file and re-displayed on the screen.
- i. The User selects a file in the list of files within the SSAP and clicks *Delete*.

- j. The access permissions of the User for the Delete File capability are checked. It is assumed that the User has permission to delete a file from the SSAP. The fact that the user is deleting a file from the SSAP is recorded in the Activity Log. If the user does not have permission to delete files from an SSAP an error message is displayed to the screen and step (k) is not done.
- k. The selected file is deleted from the SSAP. The list of files in the SSAP is updated to remove the deleted file and is re-displayed on the screen.
- l. User exits the Edit SSAP File List GUI.
- m. The list of SSAPs stored at the Data Server is refreshed on the SSAP GUI. The SSAP GUI awaits the next user selection. The User is now free to make other changes to the SSAP or to submit it to the Data Server for archiving.

#### **6.5.3.2.6 Event Trace**

Figure 6.5-8 shows the adding and deleting of files from an SSAP.



**Figure 6.5-8. Edit the File List of a Science Software Archive Package**

### **6.5.3.3 Editing the Metadata of the Chosen Science Software Archive Package**

#### **6.5.3.3.1 Abstract**

This scenario describes the manipulation of the metadata of an SSAP via the Edit SSAP Metadata GUI.

#### **6.5.3.3.2 Stimulus**

DAAC Operations Staff want to edit the metadata of an SSAP. After starting the SSAP GUI (described in Section 6.5.3.1), the staff selects the SSAP from the displayed list, and chooses the menu choice to Edit SSAP Metadata. Choices are made on the Edit SSAP Metadata GUI to alter the metadata of the chosen SSAP.

#### **6.5.3.3.3 Desired Response**

The chosen fields in the SSAP metadata are altered.

#### **6.5.3.3.4 Participating Classes from the Object Model**

- DpAtSSAPGuiNB
- DpAtEditSSAPMetaDataGuiNB
- DpAtAccessNB

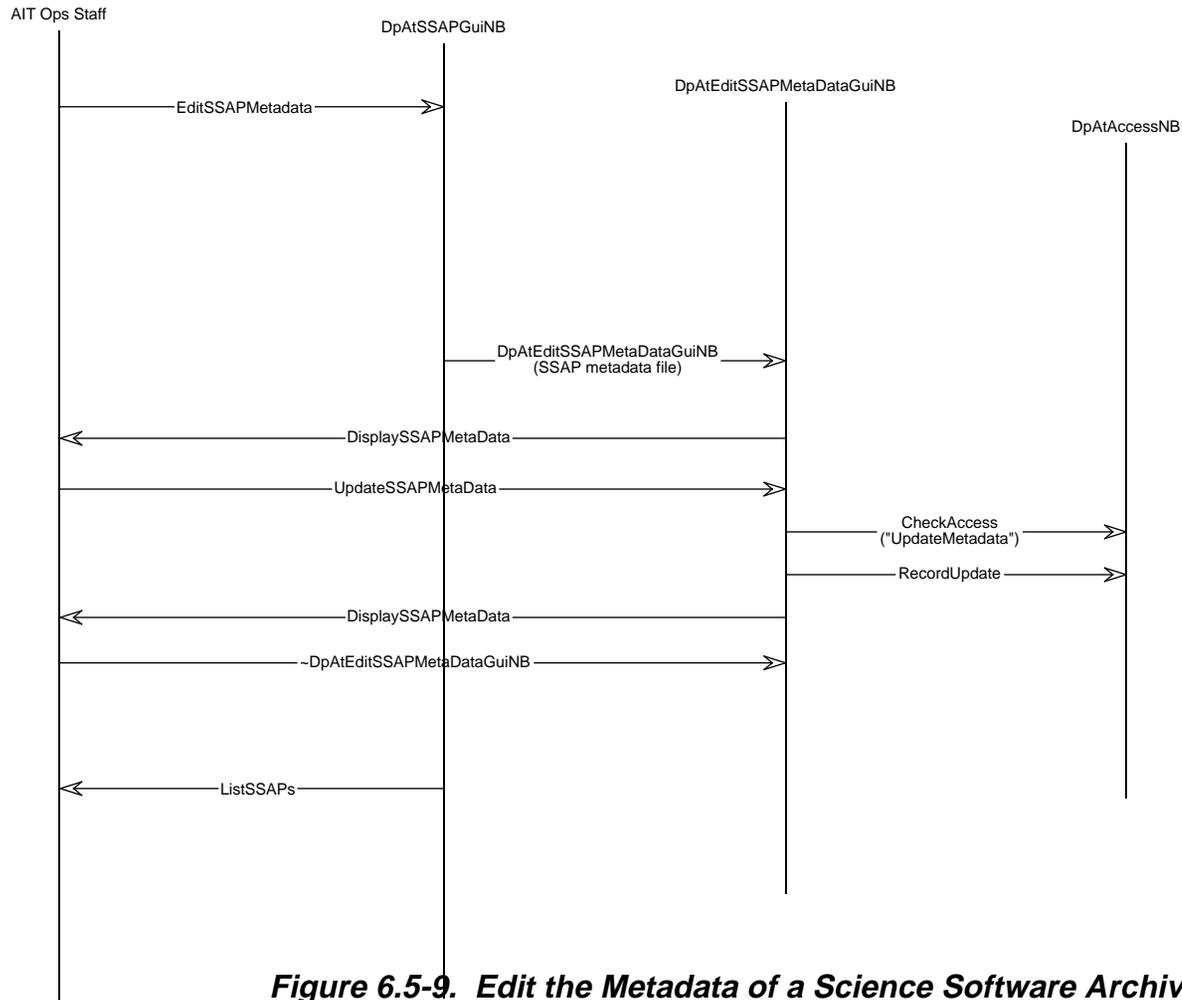
#### **6.5.3.3.5 Scenario Description**

- a. Science Software Archive Package GUI has been previously displayed, complete with a list of Science Software Archive Packages stored at the Data Server.
- b. User selects one of the SSAPs.
- c. User clicks on *Metadata*.
- d. If the SSAP has not been previously retrieved (not shown), an ACQUIRE command is formed and sent to the Data Server to retrieve the selected SSAP. The SSAP is placed on the local processor by the Data Server.
- e. Science Software Archive Package Metadata GUI is displayed. The metadata is displayed in parameter=value format.
- f. User selects the parameter or value that is to be changed. User enters new value and clicks *Save*.
- g. The access permissions of the User for the Update Metadata capability are checked. It is assumed that the User has permission to update SSAP Metadata. The fact that the user is altering the metadata of the SSAP is recorded in the Activity Log. If the user does not have permission to alter the metadata of an SSAP an error message is displayed to the screen and step (h) is not done.
- h. The change to the metadata is made in the metadata file that accompanies the SSAP. The metadata is re-displayed to show the change.
- i. User exits the Edit SSAP Metadata GUI.

- j. The list of SSAPs stored at the Data Server is refreshed on the SSAP GUI. The SSAP GUI awaits the next user selection. The User is now free to make other changes to the SSAP or to submit it to the Data Server for archiving.

#### **6.5.3.3.6 Event Trace**

Figure 6.5-9 shows the updating of the SSAP metadata.



Data Server access is not shown. If the SSAP had not yet been retrieved from the Data Server, it would have to be Acquired as in the EditSSAPFileList event trace.

**Figure 6.5-9. Edit the Metadata of a Science Software Archive Package**

## 6.5.3.4 Creating a new Science Software Archive Package

### 6.5.3.4.1 Abstract

This scenario describes the creation of a new Science Software Archive Package.

### 6.5.3.4.2 Stimulus

DAAC Operations Staff want to create a new SSAP. After starting the SSAP GUI (described in Section 6.5.3.1), the staff selects an SSAP (to copy into the new SSAP) from the displayed list, and clicks *Create*.

### 6.5.3.4.3 Desired Response

A new SSAP is created that can be manipulated by the user.

### 6.5.3.4.4 Participating Classes from the Object Model

- DpAtSSAPGuiNB
- DpAtEditSSAPFileListGuiNB
- DpAtAccessNB

### 6.5.3.4.5 Scenario Description

- a. Science Software Archive Package GUI has been previously displayed, complete with a list of Science Software Archive Packages stored at the Data Server.
- b. User selects one of the SSAPs. Note that the User does not have to select an SSAP to copy from. If no SSAP is selected when the User clicks *Create*, step (f) is skipped and an SSAP with an empty file list is displayed in step (g).
- c. User clicks on *Create*.
- d. The User is queried for the name of the new SSAP. The User enters the name and clicks *OK*.
- e. The access permissions of the User for the Create SSAP capability are checked. It is assumed that the User has permission to create an SSAP. The fact that the user is creating an SSAP is recorded in the Activity Log. If the user does not have permission to create a new SSAP an error message is displayed to the screen and the rest of the steps are not done.
- f. An ACQUIRE command is formed and sent to the Data Server to retrieve the selected SSAP. The SSAP is placed on the local processor by the Data Server. The contents of this SSAP (files, metadata) are copied to the new SSAP being created.
- g. Science Software Archive Package File List GUI is displayed to allow the user to manipulate the files in the new SSAP. A list of all of the files contained in the SSAP is placed on the screen.
- h. User updates the *Directory Location* for the files to Add/Replace and selects one of the files listed in the window. User then clicks *Add*.

- i. The access permissions of the User for the Add File capability are checked. It is assumed that the User has permission to add a file to the SSAP. The fact that the user is adding a file to the SSAP is recorded in the Activity Log. If the user does not have permission to add files to an SSAP an error message is displayed to the screen and step (j) is not done.
- j. The selected file is added to the SSAP. The list of files in the SSAP is updated to include the added file and is re-displayed to the screen.
- k. The User selects a file in the list of files within the SSAP and clicks *Delete*.
- l. The access permissions of the User for the Delete File capability are checked. It is assumed that the User has permission to delete a file from the SSAP. The fact that the user is deleting a file from the SSAP is recorded in the Activity Log. If the user does not have permission to delete files from an SSAP an error message is displayed to the screen and step (m) is not done.
- m. The selected file is deleted from the SSAP. The list of files in the SSAP is updated to remove the deleted file and is re-displayed to the screen.
- n. User exits the Edit SSAP File List GUI.
- o. The list of SSAPs stored at the Data Server is refreshed. The SSAP GUI awaits the next user selection. The User is now free to make other changes to the new SSAP or to submit it to the Data Server for archiving.

#### **6.5.3.4.6 Event Trace**

Figure 6.5-10 shows the creation of an SSAP.

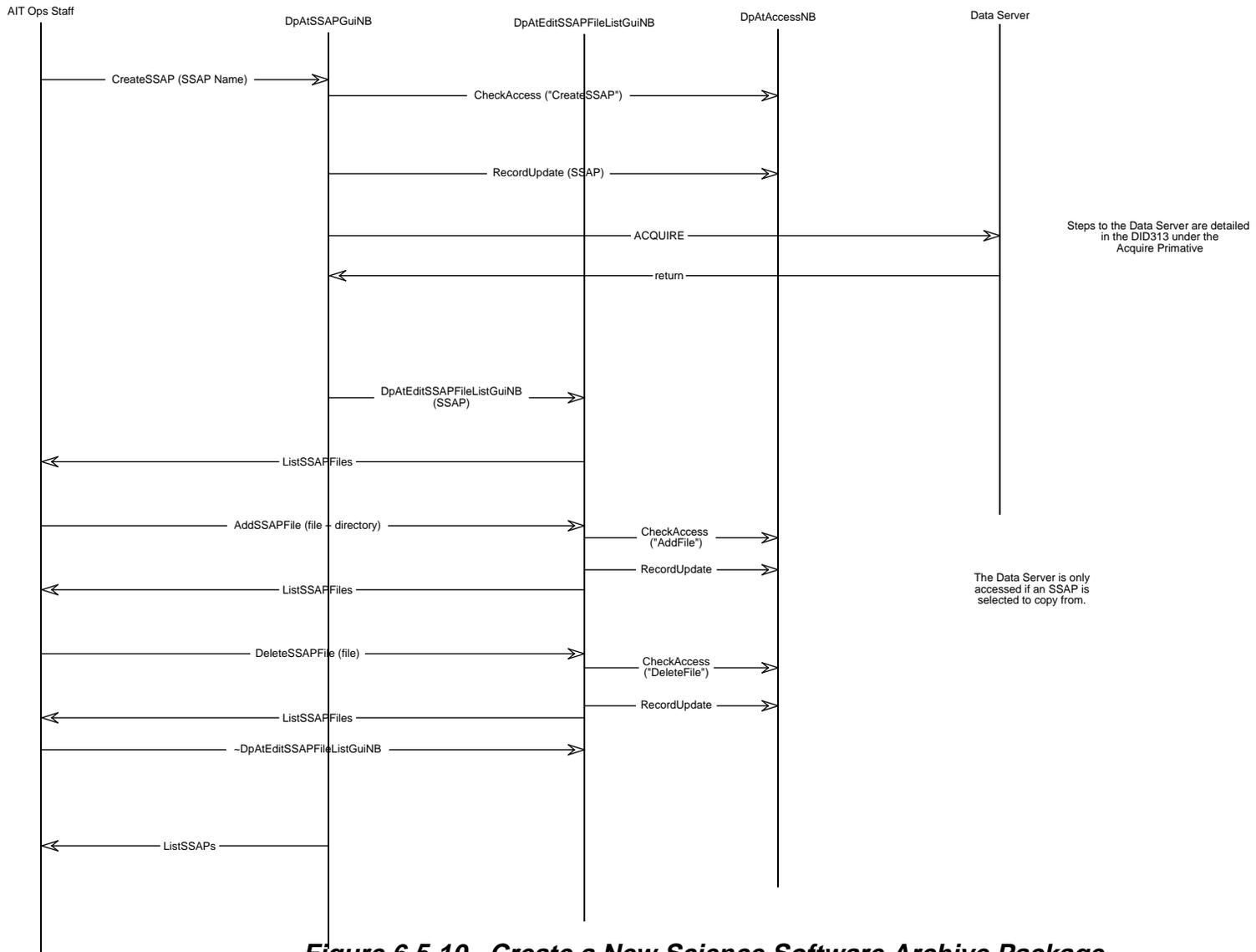


Figure 6.5-10. Create a New Science Software Archive Package

### **6.5.3.5 Deleting a Science Software Archive Package**

#### **6.5.3.5.1 Abstract**

This scenario describes the deletion of a Science Software Archive Package.

#### **6.5.3.5.2 Stimulus**

DAAC Operations Staff want to delete an existing SSAP. After starting the SSAP GUI (described in Section 6.5.3.1), the staff selects the SSAP from the list of SSAPs on the Data Server and clicks *Delete*.

#### **6.5.3.5.3 Desired Response**

The SSAP is deleted.

#### **6.5.3.5.4 Participating Classes from the Object Model**

- DpAtSSAPGuiNB
- DpAtAccessNB
- DsCICommand
- DsCIRequest
- GIParameter
- GIParameterList

#### **6.5.3.5.5 Scenario Description**

- a. Science Software Archive Package GUI has been previously displayed, complete with a list of Science Software Archive Packages stored at the Data Server.
- b. User selects one of the SSAPs.
- c. User clicks on *Delete*.
- d. The access permissions of the User for the Delete SSAP capability are checked. It is assumed that the User has permission to delete an SSAP. The fact that the user is deleting an SSAP is recorded in the Activity Log. If the user does not have permission to delete an SSAP an error message is displayed to the screen and step (e) is not done.
- e. A DELETE command is formed and sent to the Data Server to remove the selected SSAP.
- f. The list of SSAPs stored at the Data Server is refreshed so that the deleted SSAP is no longer shown. The SSAP GUI awaits the next user selection. The User is now free to make changes to other SSAPs.

#### **6.5.3.5.6 Event Trace**

Figure 6.5-11 shows the creation of an SSAP.

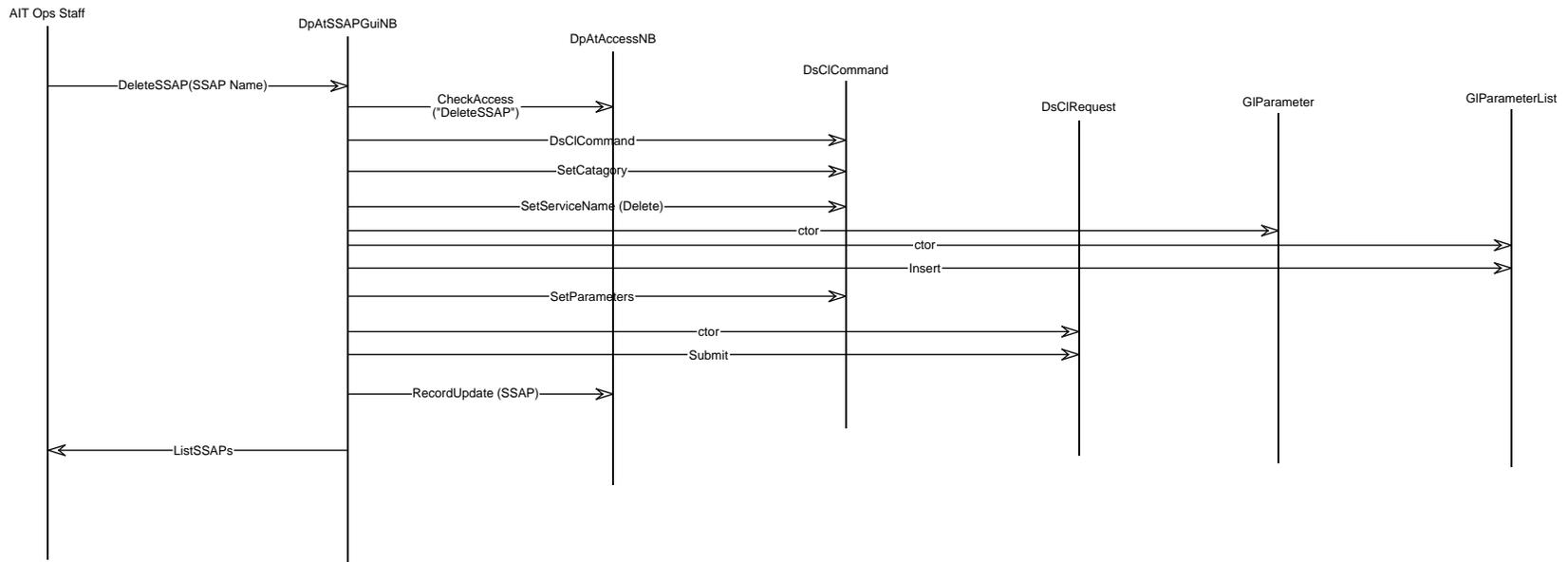


Figure 6.5-11. Delete a Science Software Archive Package

### **6.5.3.6 Submitting a changed Science Software Archive Package to the Data Server**

#### **6.5.3.6.1 Abstract**

This scenario describes the Insertion of a changed SSAP to the Data Server. This could either be a previously existing SSAP, or a new SSAP created by the Operations Staff.

#### **6.5.3.6.2 Stimulus**

DAAC Operations Staff have finished their changes to the current SSAP and want it stored back on the Data Server. The staff clicks *Submit*.

#### **6.5.3.6.3 Desired Response**

The SSAP is archived to the Data Server.

#### **6.5.3.6.4 Participating Classes from the Object Model**

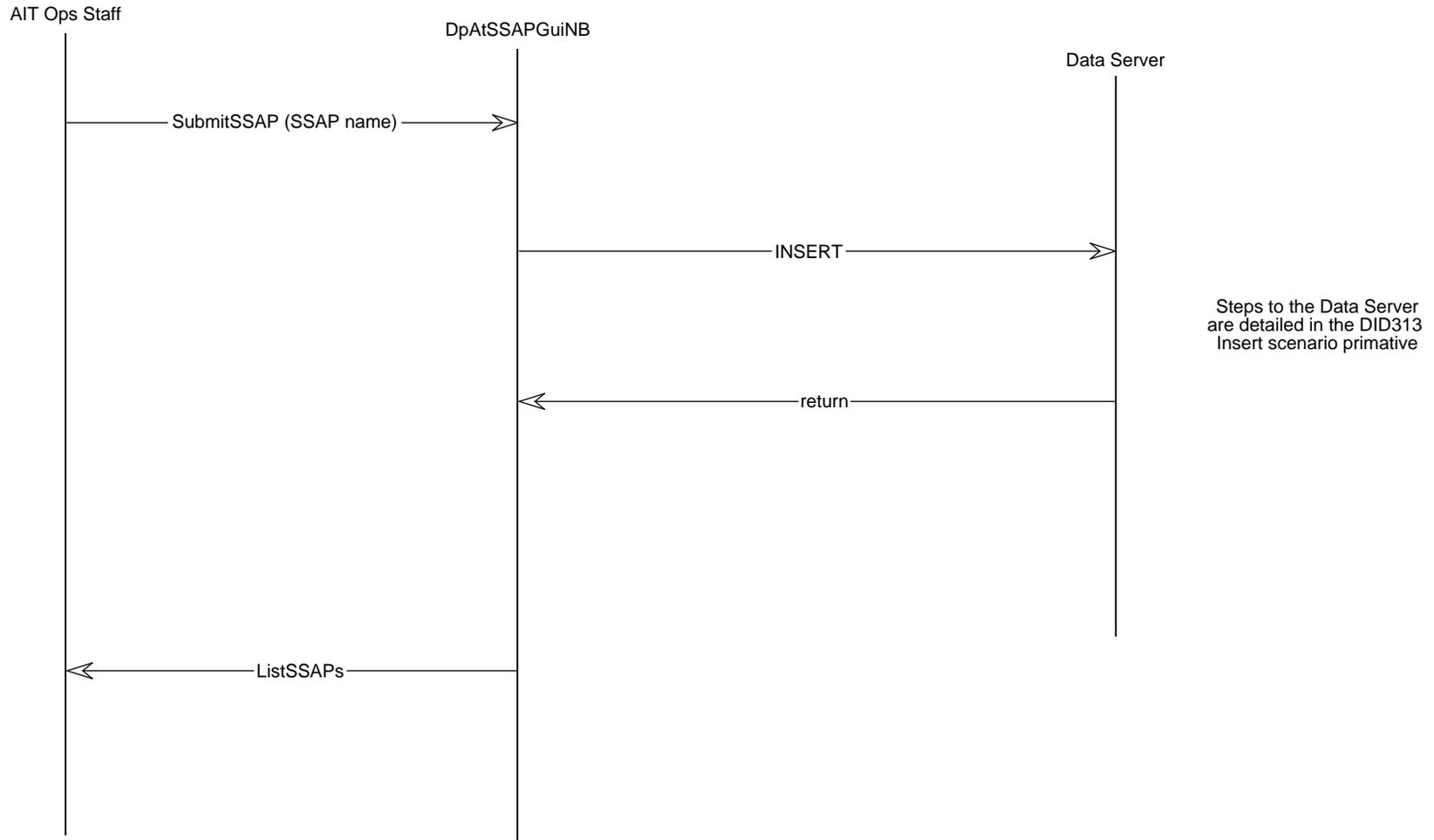
- DpAtSSAPGuiNB

#### **6.5.3.6.5 Scenario Description**

- a. Science Software Archive Package GUI has been previously displayed, complete with a list of Science Software Archive Packages stored at the Data Server. Changes have been made to the currently selected SSAP.
- b. User clicks on *Submit*.
- c. An INSERT command is formed and sent to the Data Server to insert the changed SSAP.
- d. The list of SSAPs stored at the Data Server is refreshed so that the changed SSAP shows the updated version. The SSAP GUI awaits the next user selection. The User is now free to make changes to other SSAPs.

#### **6.5.3.6.6 Event Trace**

Figure 6.5-12 shows the submission of an SSAP.



**Figure 6.5-12. Submit an Updated Science Software Archive Package to the Data Server**

## 6.5.4 PGE Registration Scenarios

This section describes scenarios for the Registration of a PGE in the PDPS database via the AITTL PGE Registration GUIs.

### 6.5.4.1 Updating the Activation Rule Information for an Existing Time Scheduled PGE

#### 6.5.4.1.1 Abstract

This scenario describes the altering of a Time Scheduled PGE's activation rule information through the PGE Activation Rule GUI.

#### 6.5.4.1.2 Stimulus

DAAC Operations Staff want to update a Time Scheduled PGE's activation rule information. The staff chooses the PGE Activation Rule GUI option from the PGE Registration GUI.

#### 6.5.4.1.3 Desired Response

The PGE activation rule attributes are changed and stored in the PDPS database.

#### 6.5.4.1.4 Participating Classes from the Object Model

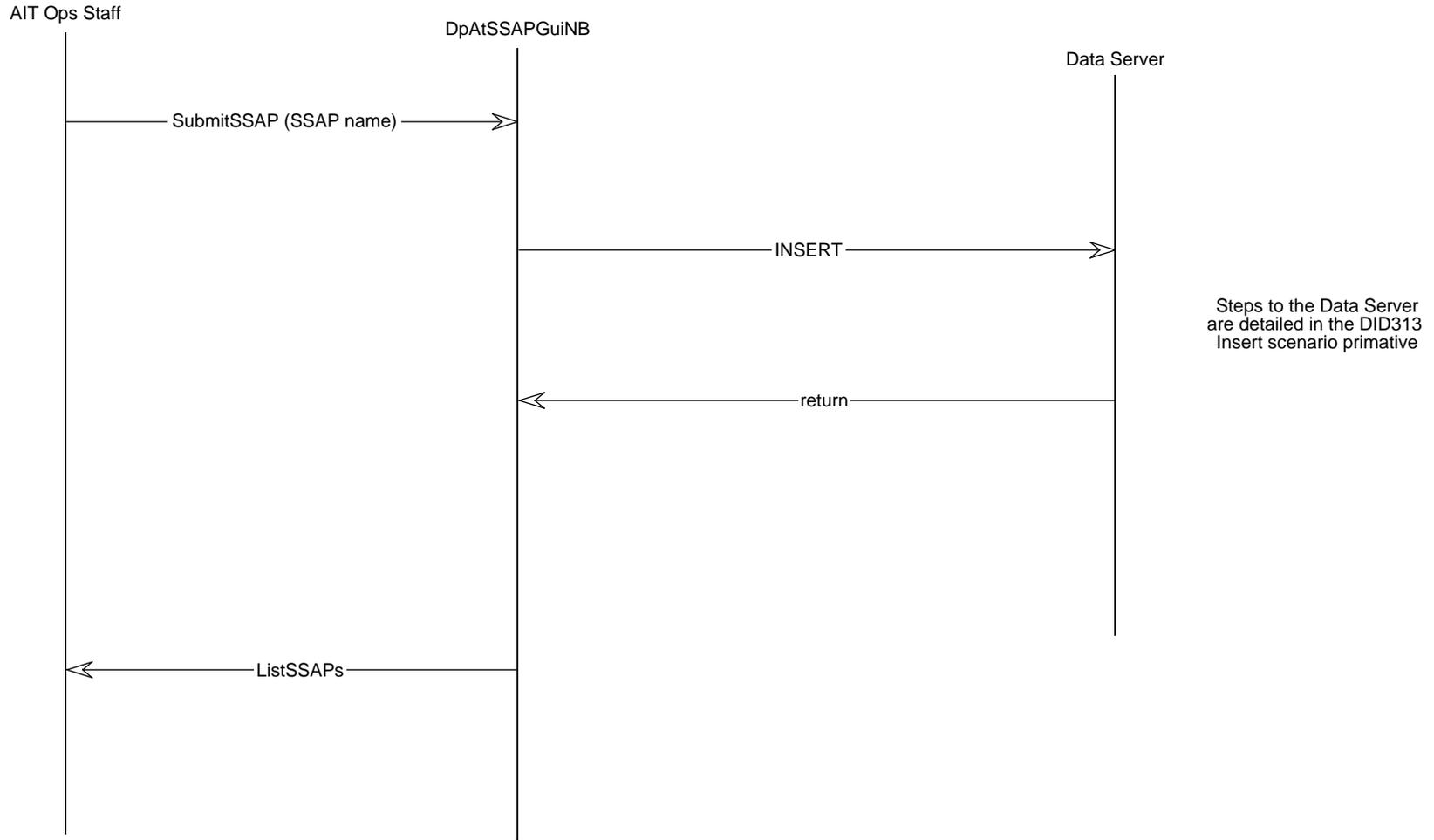
- DpAtPgeRegistrationGui
- DpAtPgeActivationRuleB
- PTimeScheduled
- DpPrDbInterface

#### 6.5.4.1.5 Scenario Description

- a. PGE Registration GUI has been previously displayed.
- b. User clicks *Activation Rule*.
- c. PGE Activation Rule GUI is displayed. The Time Scheduled attributes for the current PGE are retrieved from the PDPS database via the class DpPrDbInterface. The attributes and their values are displayed to the screen.
- d. User chooses an attribute to modify and enters the new value. User clicks *OK*.
- e. The change is made to the local copy of the Time Scheduled attributes and the update is displayed to the screen.
- f. User clicks on *Submit*.
- g. The change to the Time Scheduled attribute is stored in the PDPS database via the class DpPrDbInterface.
- h. User exits the PGE Activation Rule GUI and is returned to the PGE Registration GUI.

#### 6.5.4.1.6 Event Trace

Figure 6.5-13 shows updating an existing Time Scheduled PGE's activation rule information.



**Figure 6.5-13. Update an Exiting Time Scheduled PGE's Profile**

## 6.5.4.2 Updating the Activation Rule Information for an Existing Tile Scheduled PGE

### 6.5.4.2.1 Abstract

This scenario describes the altering of a Tile Scheduled PGE's activation rule information and corresponding Tile Clustering scheme information, through the PGE Activation Rule GUI.

### 6.5.4.2.2 Stimulus

DAAC Operations Staff want to update a Tile Scheduled PGE's activation rule information. The staff chooses the PGE Activation Rule GUI option from the PGE Registration GUI.

### 6.5.4.2.3 Desired Response

The PGE activation rule attributes, and the corresponding Tile Cluster definition, are changed and stored in the PDPS database.

### 6.5.4.2.4 Participating Classes from the Object Model

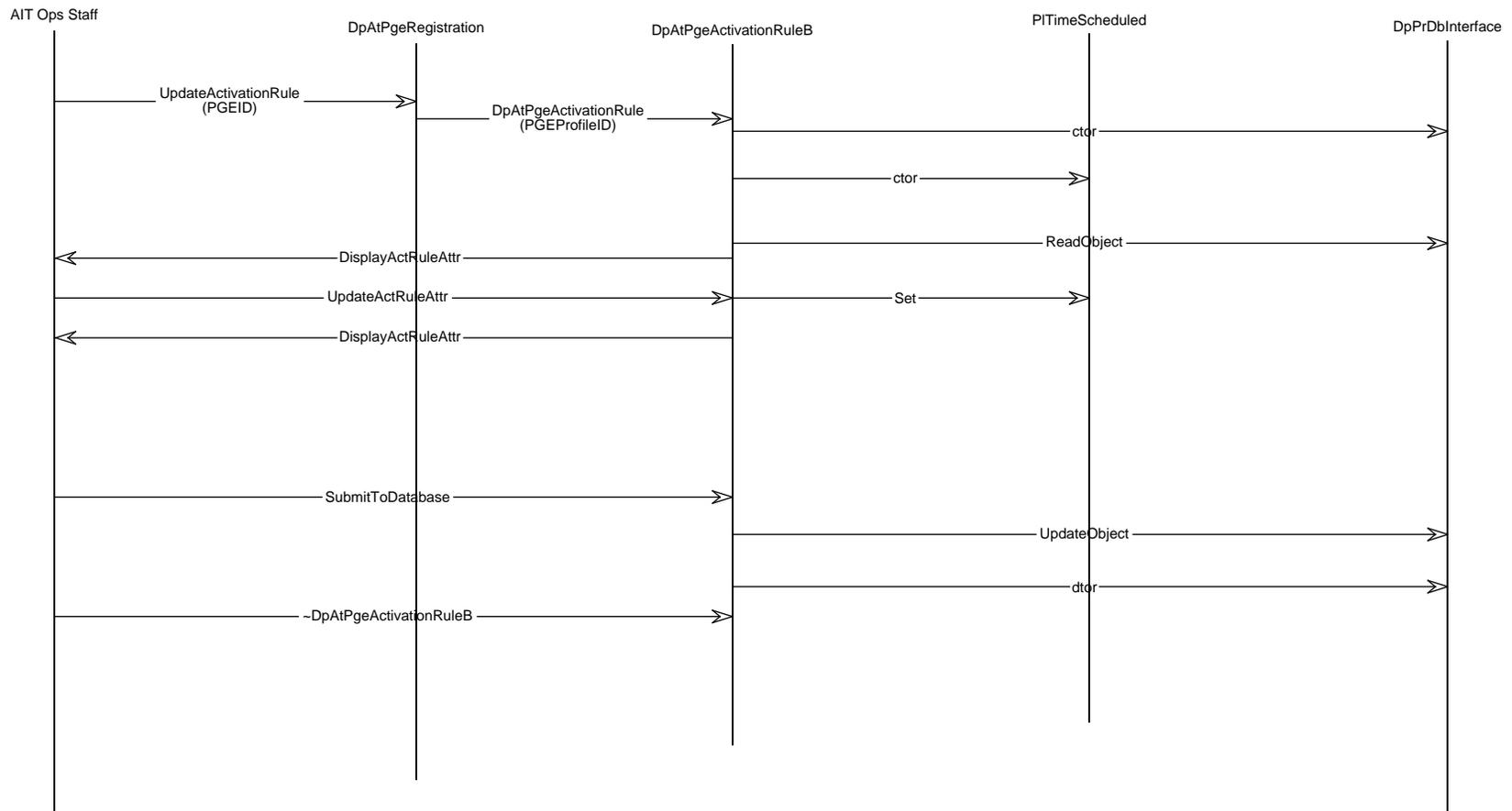
- DpAtPgeRegistrationGui
- DpAtPgeActivationRuleB
- PTileScheduledNB
- PCluster
- DpPrDbInterface

### 6.5.4.2.5 Scenario Description

- a. PGE Registration GUI has been previously displayed.
- b. User clicks *Activation Rule*.
- c. PGE Activation Rule GUI is displayed. The Tile Scheduled attributes, plus the PCluster attributes, for the current PGE are retrieved from the PDPS database via the class DpPrDbInterface. The attributes and their corresponding values are displayed to the screen.
- d. User chooses a Tile Scheduled attribute to modify and enters the new value. User clicks *OK*.
- e. The change is made to the local copy of the Tile Scheduled attributes and the update is displayed to the screen.
- f. User chooses a Cluster attribute to modify and enters a new value. User clicks *OK*.
- g. The change is made to the local copy of the Cluster and the update is displayed on the screen.
- h. User clicks on *Submit*.
- i. The changes to the Tile Scheduled information and the Cluster are stored in the PDPS database via the class DpPrDbInterface.
- j. User exits the PGE Activation Rule GUI and is returned to the PGE Registration GUI.

### 6.5.4.2.6 Event Trace

Figure 6.5-14 shows updating an existing Time Scheduled PGE's activation rule information.



**Figure 6.5-14. Update an Existing Tile Scheduled PGE's Profile**

### **6.5.4.3 Creating the Activation Rule Information for a new Data Scheduled PGE**

#### **6.5.4.3.1 Abstract**

This scenario describes the creation of a Data Scheduled PGE's activation rule information, through the PGE Activation Rule GUI.

#### **6.5.4.3.2 Stimulus**

DAAC Operations Staff want to create a Data Scheduled PGE's activation rule information. The staff chooses the PGE Activation Rule GUI option from the PGE Registration GUI.

#### **6.5.4.3.3 Desired Response**

The PGE activation rule attributes are created and stored in the PDPS database.

#### **6.5.4.3.4 Participating Classes from the Object Model**

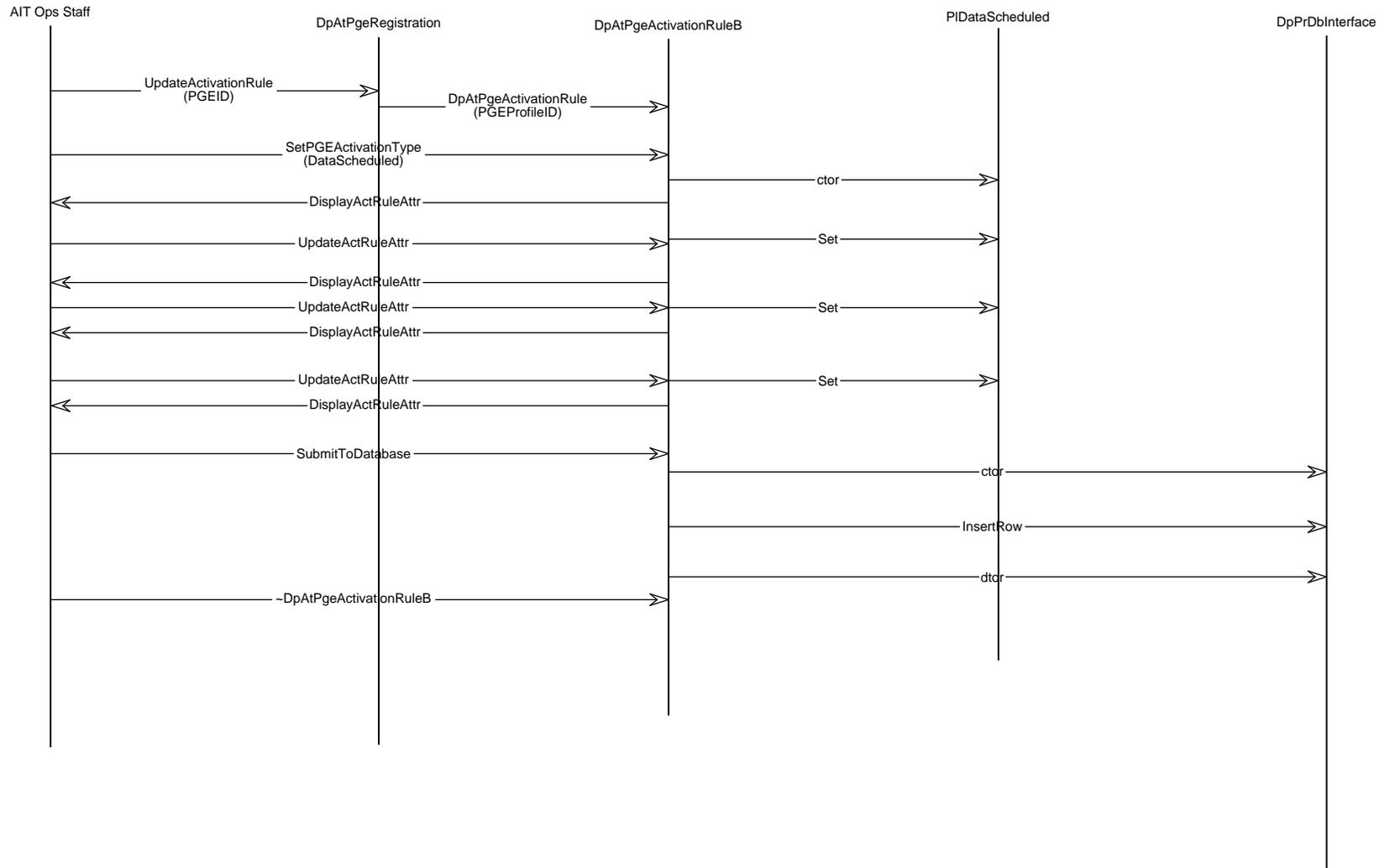
- DpAtPgeRegistrationGui
- DpAtPgeActivationRuleB
- PIDataScheduled
- DpPrDbInterface

#### **6.5.4.3.5 Scenario Description**

- a. PGE Registration GUI has been previously displayed.
- b. User clicks *Activation Rule*.
- c. PGE Activation Rule GUI is displayed. Since the PGE is new, no attributes are displayed because the activation rule type of the PGE is not known.
- d. User selects Data Scheduled for the activation rule and clicks *OK*.
- e. The Data Scheduled attributes are displayed to the screen with empty (null) values.
- f. User chooses each Data Scheduled attribute to modify and enters the new value. User clicks *OK* after each update.
- g. Each change is made to the local copy of the Data Scheduled attributes and the update is displayed to the screen.
- h. User clicks on *Submit*.
- i. The new Data Scheduled information is stored in the PDPS database via the class DpPrDbInterface.
- j. User exits the PGE Activation Rule GUI and is returned to the PGE Registration GUI.

#### **6.5.4.3.6 Event Trace**

Figure 6.5-15 shows creating a new Data Scheduled PGE's activation rule information.



**Figure 6.5-15. Create a New Data Scheduled PGE's Activation Rule**

## 6.5.4.4 Creating the Activation Rule Information for a new Orbit Scheduled PGE

### 6.5.4.4.1 Abstract

This scenario describes the creation of an Orbit Scheduled PGE's activation rule information and the corresponding Orbit Model, through the PGE Activation Rule GUI.

### 6.5.4.4.2 Stimulus

DAAC Operations Staff want to create an Orbit Scheduled PGE's activation rule information. The staff chooses the PGE Activation Rule GUI option from the PGE Registration GUI.

### 6.5.4.4.3 Desired Response

The PGE activation rule attributes, and the corresponding Tile Cluster definition, are changed and stored in the PDPS database.

### 6.5.4.4.4 Participating Classes from the Object Model

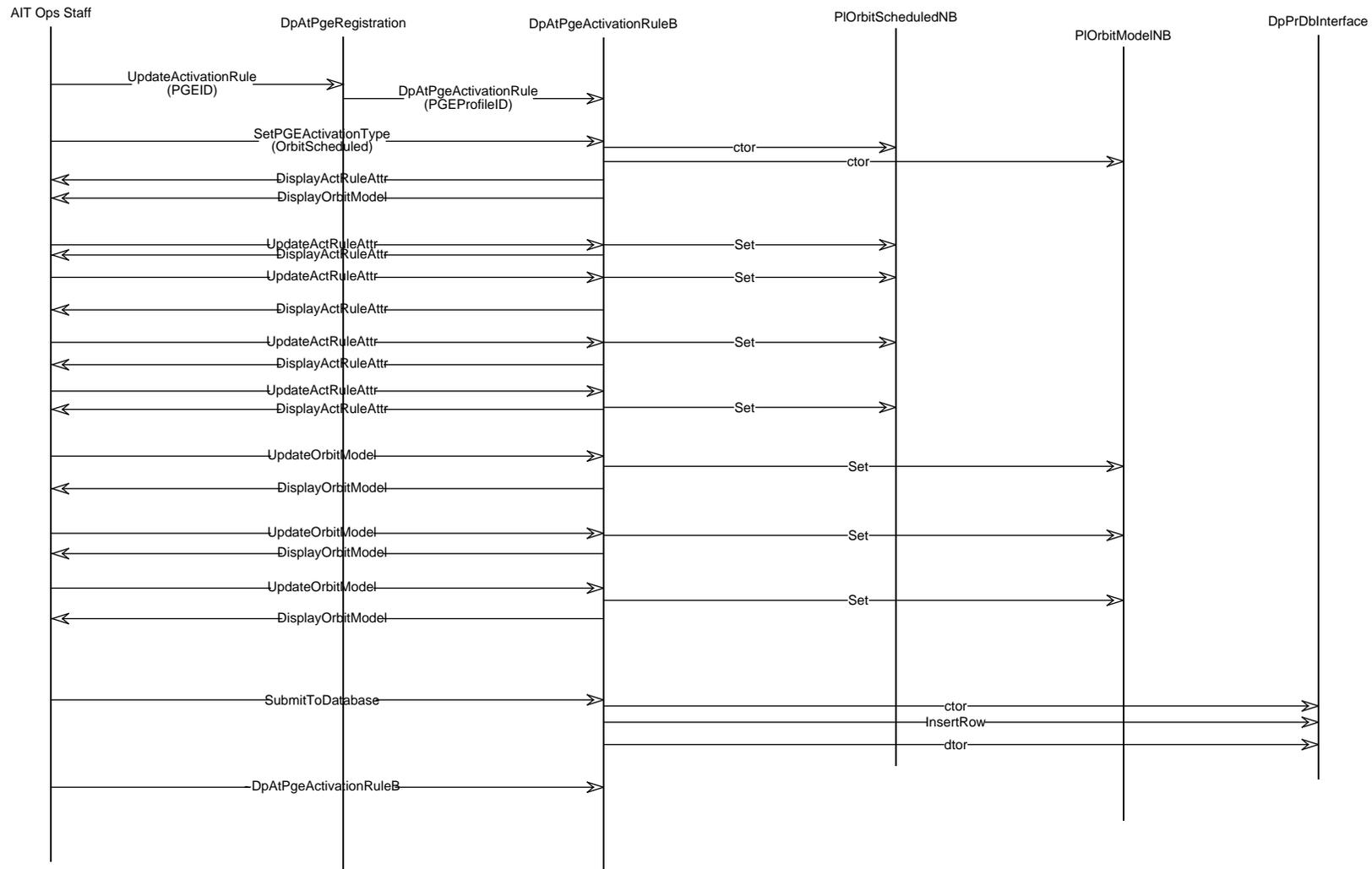
- DpAtPgeRegistrationGui
- DpAtPgeActivationRuleB
- PLOrbitScheduledNB
- PLOrbitModelNB
- DpPrDbInterface

### 6.5.4.4.5 Scenario Description

- a. PGE Registration GUI has been previously displayed.
- b. User clicks *Activation Rule*.
- c. PGE Activation Rule GUI is displayed. Since the PGE is new, no attributes are displayed because the activation rule type of the PGE is not known.
- d. User selects Orbit Scheduled for the activation rule and clicks *OK*.
- e. The Orbit Scheduled attributes are displayed to the screen with empty (null) values. An empty Orbit Model (attributes with null values) is also displayed.
- f. User chooses each Orbit Scheduled attribute to modify and enters the new value. User clicks *OK* after each update.
- g. Each change is made to the local copy of the Orbit Scheduled attributes and the update is displayed to the screen.
- h. User chooses each Orbit Model attribute to modify and enters the new value. User clicks *OK* after each update.
- i. Each change is made to the local copy of the Orbit Model attributes and the update is displayed to the screen.
- j. User clicks on *Submit*.
- k. The new Orbit Scheduled information and the Orbit Model are stored in the PDPS database via the class DpPrDbInterface.
- l. User exits the PGE Activation Rule GUI and is returned to the PGE Registration GUI.

#### **6.5.4.4.6 Event Trace**

Figure 6.5-16 shows creating a new Orbit Scheduled PGE's activation rule information.



**Figure 6.5-16. Create a New Orbit Scheduled PGE's Activation Rule**

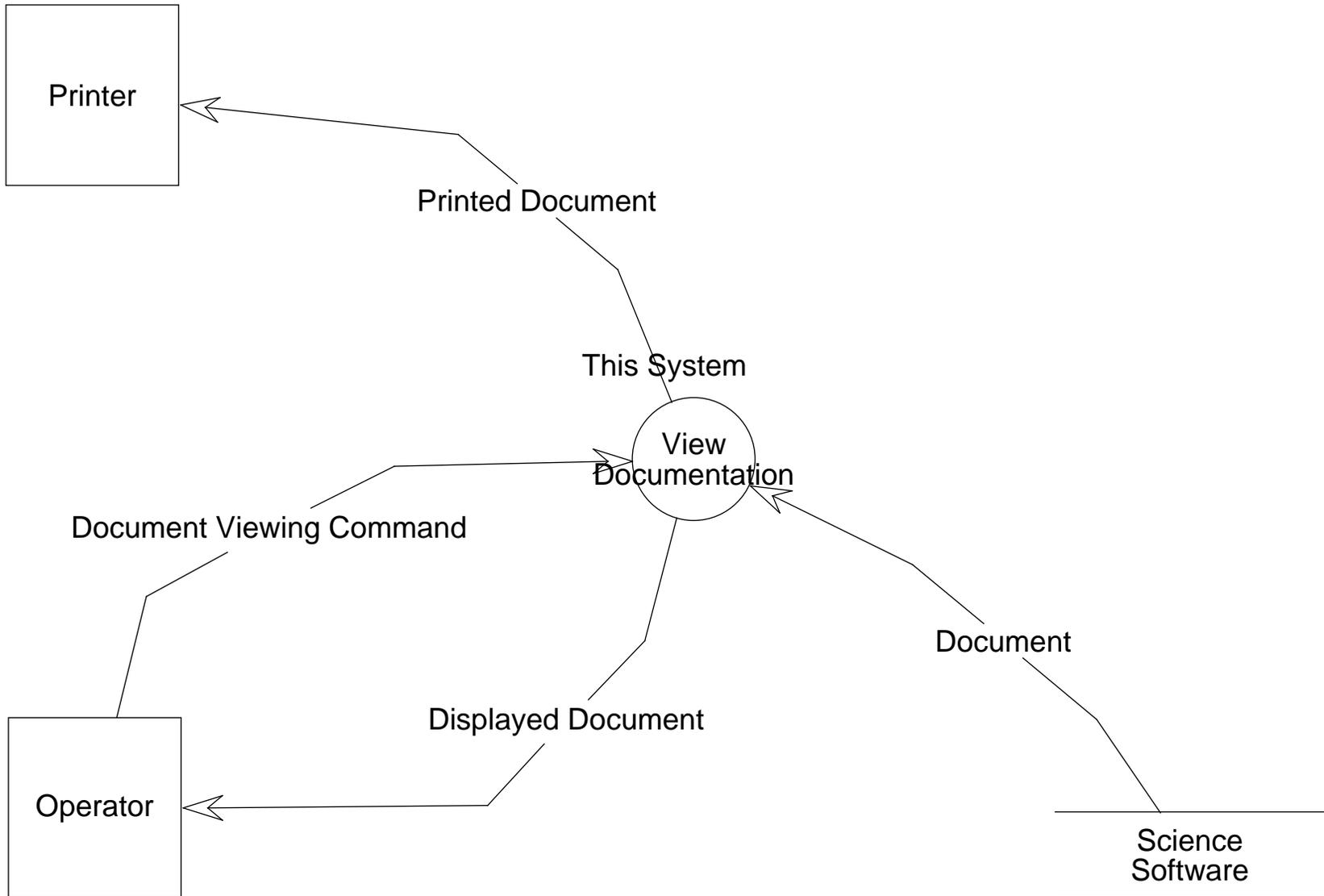
## 6.6 CSCI Functional Model

The sections which follow present a context diagram for each of the components of AITTL. These components, or tools, correspond to the computer software components (CSCs) listed below in Section 6.7, AITTL Structure.

### 6.6.1 Viewing Science Software Documentation

The context diagram for the tools to display and/or print science software documentation is shown in Figure 6.6-1.

The operator issues a command (*Document Viewing Command*) to view a particular document on the display (*Displayed Document*), or to print a hard copy (*Printed Document*) of a document. A soft copy of the document (*Document*) is stored online in the local integration and test area (*Science Software*).



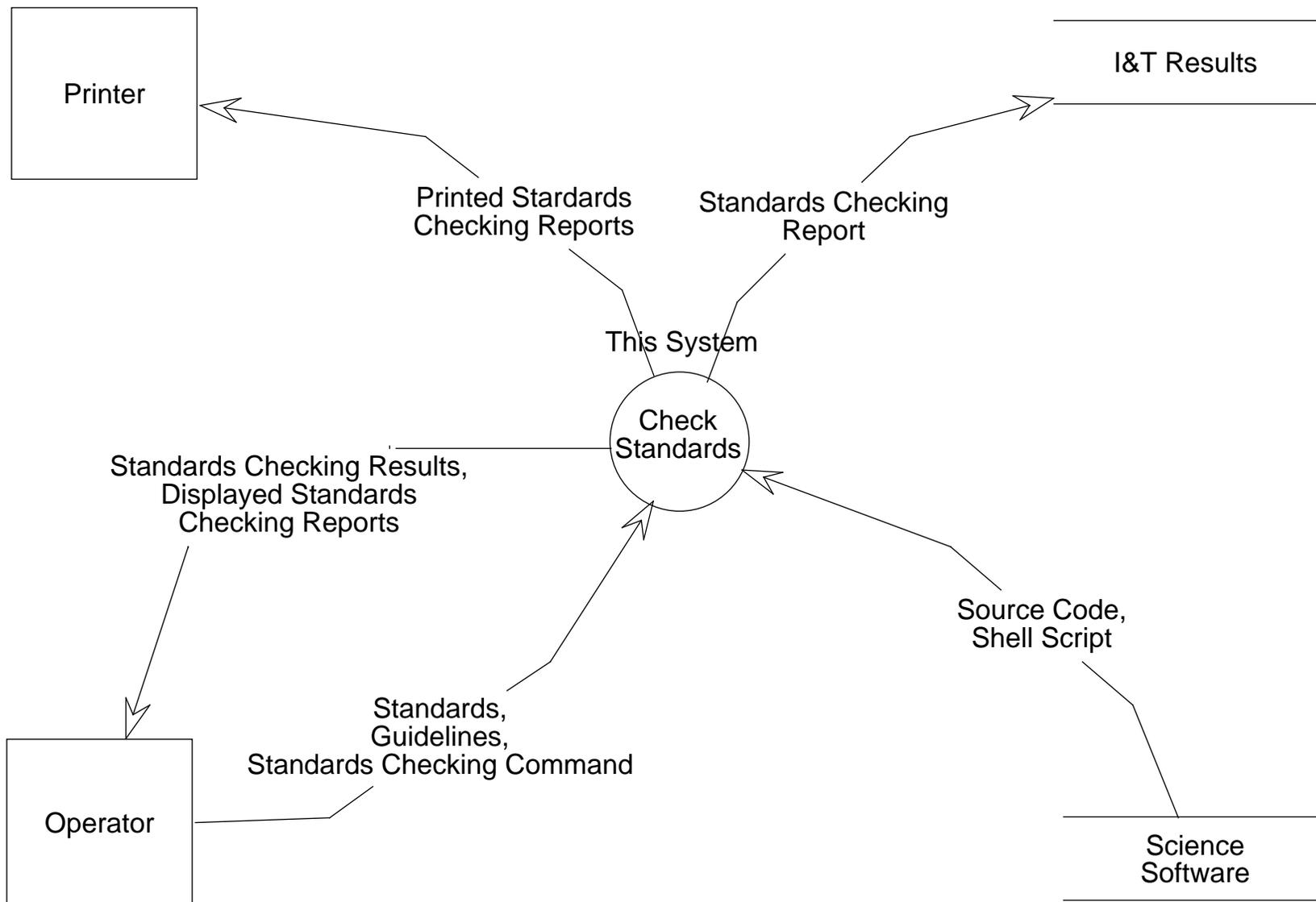
**Figure 6.6-1. View Documentation**

## 6.6.2 Checking Coding Standards

The context diagram for the standards checkers is shown in Figure 6.6-2.

The operator issues a command (*Standards Checking Command*) to check for compliance of a particular script (*Shell Script*) or source file (*Source Code*), or set of files, with certain standards and/or guidelines. The operator also must supply the standards checkers with the required standards (*Standards*) and guidelines (*Guidelines*) (only once, when the tools are configured). The results of the check (*Standards Checking Results*) are displayed on the console.

Reports may be generated as well, which may be displayed (*Displayed Standards Checking Reports*), printed (*Printed Standards Checking Reports*), and saved as soft copy (*Standards Checking Report*).

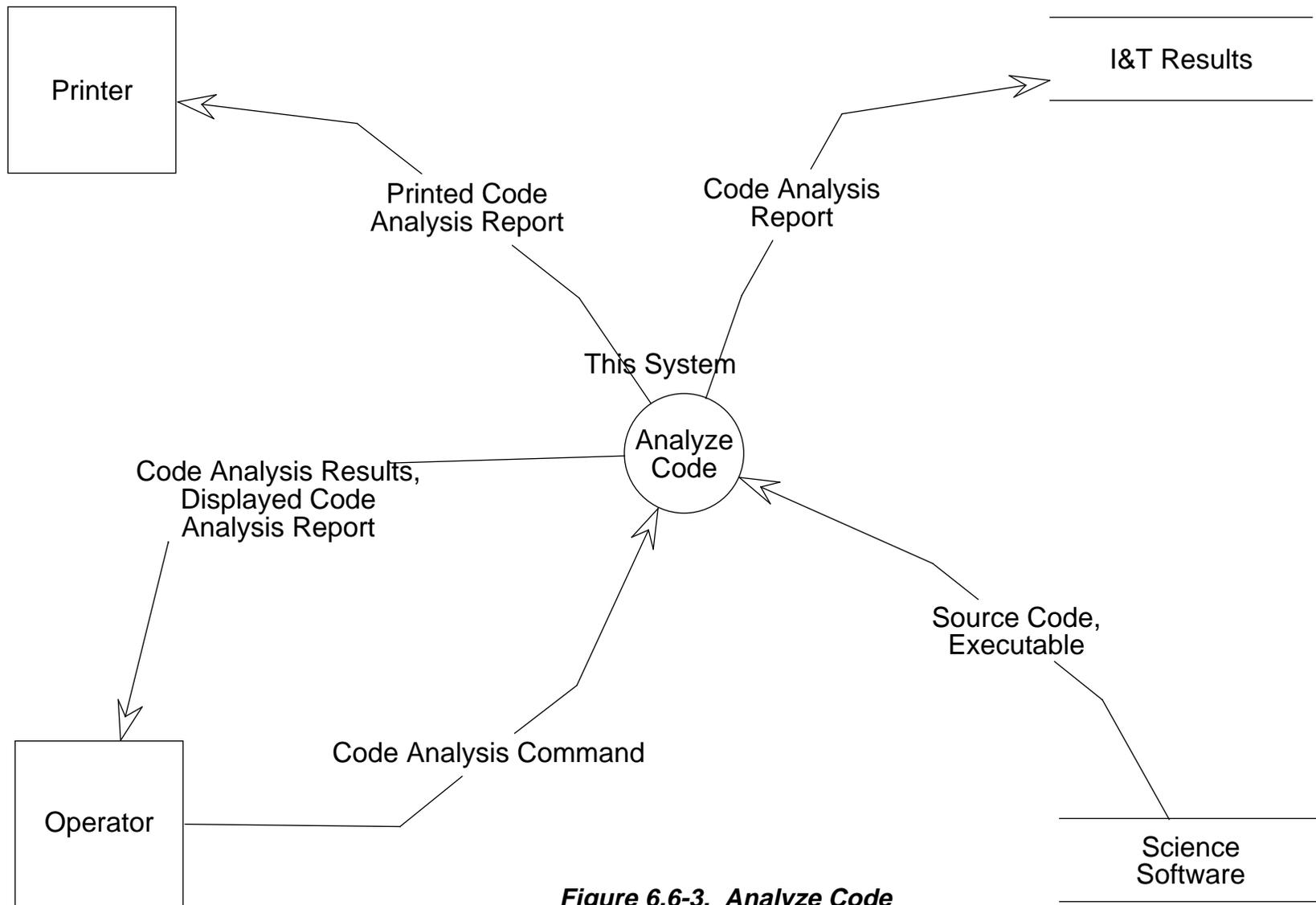


**Figure 6.6-2. Check Standards**

### 6.6.3 Analyzing the Code

The context diagram for the static and dynamic code checkers is shown in Figure 6.6-3.

The operator issues a command (*Code Analysis Command*) to make certain checks, either statistically on the source files (*Source Code*), or dynamically on the executables (*Executable*). The results (*Code Analysis Results*) of the checks are displayed on the console. Reports may also be generated, and these may be displayed (*Displayed Code Analysis Report*), printed (*Printed Code Analysis Report*), or saved as a soft copy (*Code Analysis Report*).

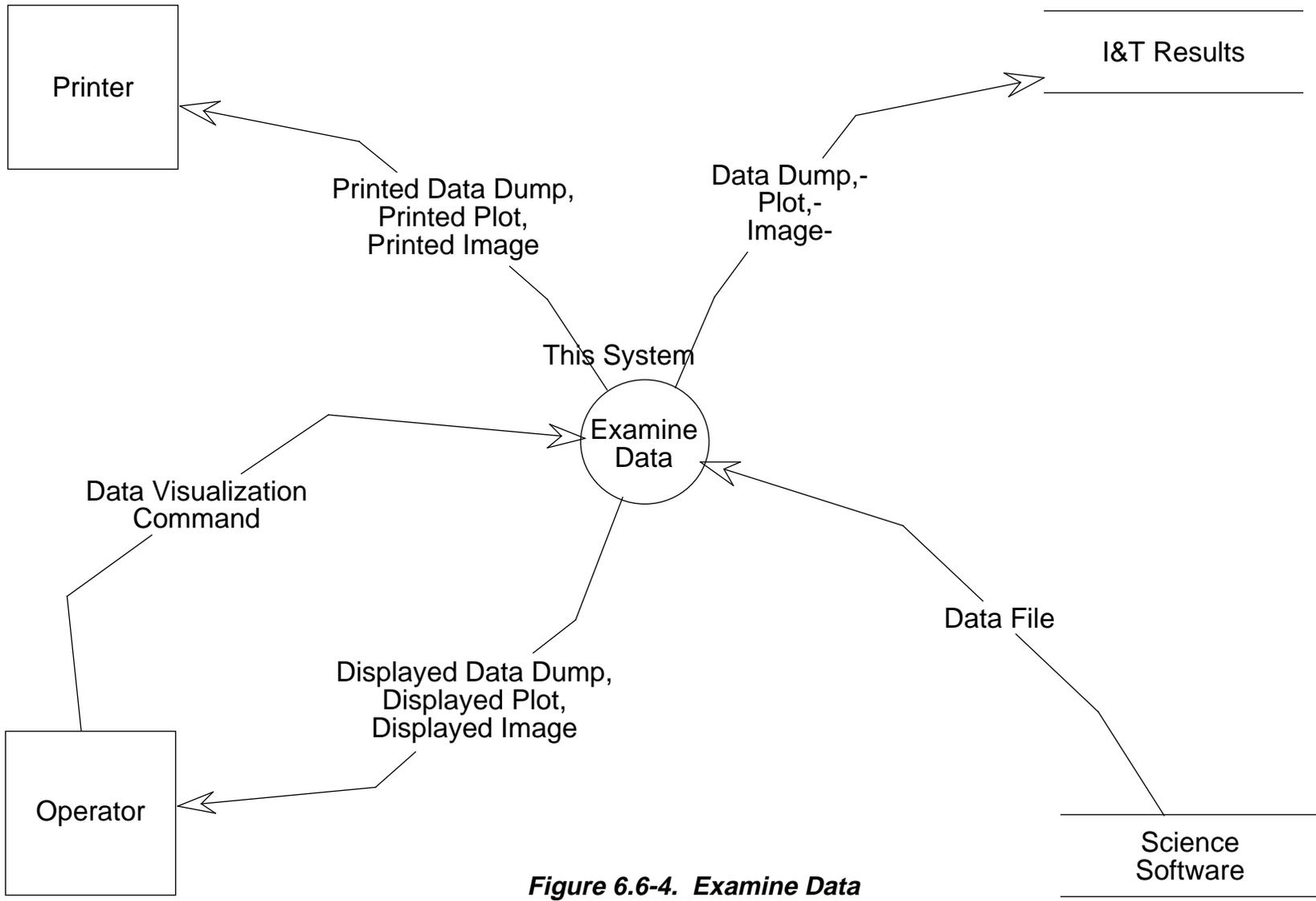


**Figure 6.6-3. Analyze Code**

#### **6.6.4 Examining the Data**

The context diagram for the data visualization tools is shown in Figure 6.5-4.

The operator issues a command (*Data Visualization Command*) to examine a particular data file. The data visualization tools may display the data in the form of a data dump (*Displayed Data Dump*), a two- or three-dimensional plot (*Displayed Plot*), or an image (*Displayed Image*). These displays may also be printed (*Printed Data Dump*, *Printed Plot*, *Printed Image*) or saved as a soft copy (*Data Dump*, *Plot*, *Image*).

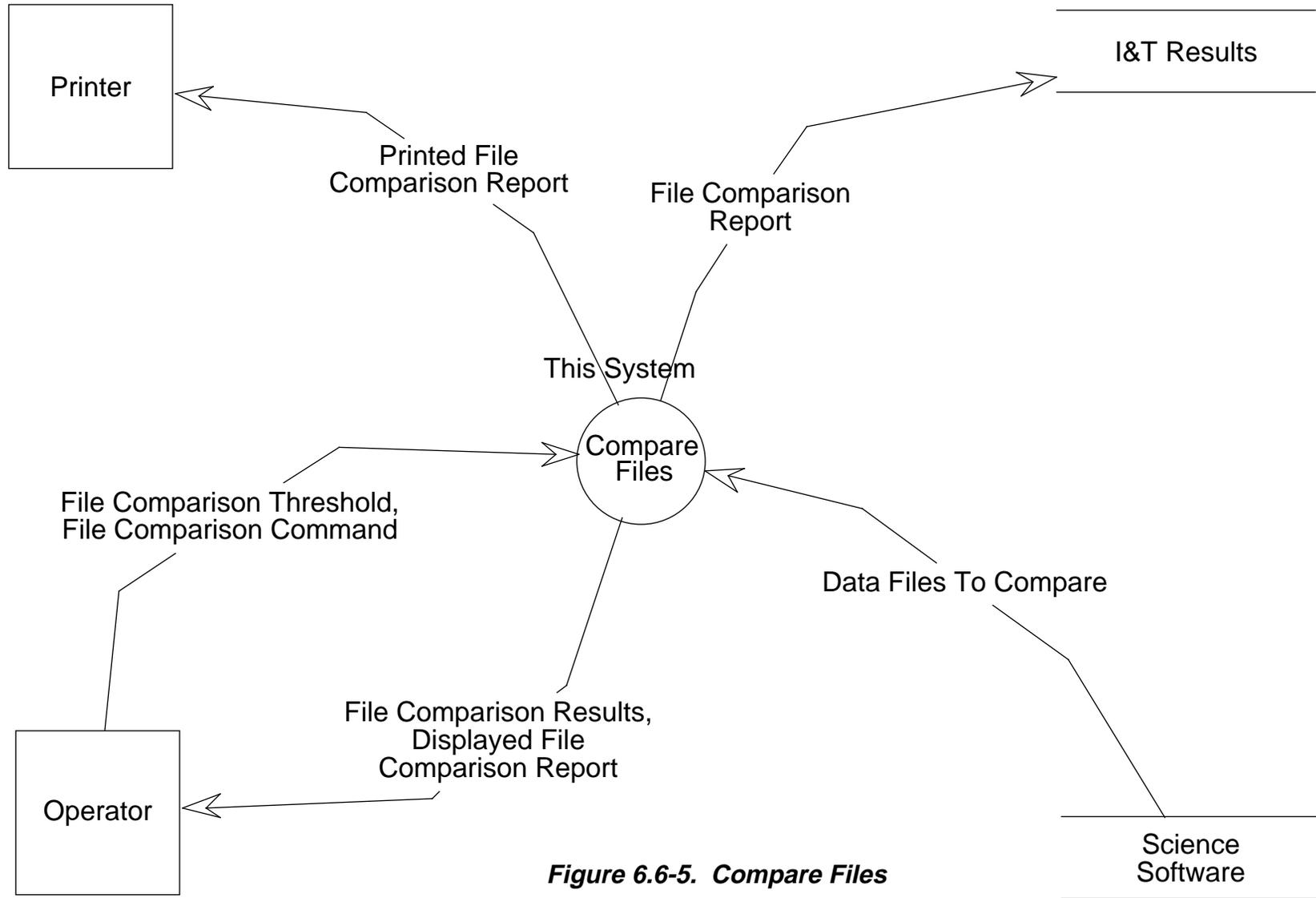


**Figure 6.6-4. Examine Data**

### 6.6.5 Comparing Data Files

The context diagram for the file comparison utility is shown in Figure 6.6-5.

The operator issues a command (*File Comparison Command*) to find all differences between two data files (*Data Files to Compare*). Normally, one of these files will have been generated by running a particular test case at the SCF, while the other will have been generated by running the same test case at the DAAC. If there are precision differences between the SCF and DAAC processing platforms, there will be corresponding differences between the two data files which should be ignored. Therefore, the operator will also supply a threshold (*File Comparison Threshold*) for masking out these types of differences. The results (*File Comparison Results*) of the file comparison are displayed on the console. Reports may also be generated, and these may be displayed (*Displayed File Comparison Report*), printed (*Printed File Comparison Report*), or saved as a soft copy (*File Comparison Report*). Please note that this section applies directly to HDF files. For binary files, the ECS supplies a programming environment which the DAAC Operator used to write custom binary file comparison code.

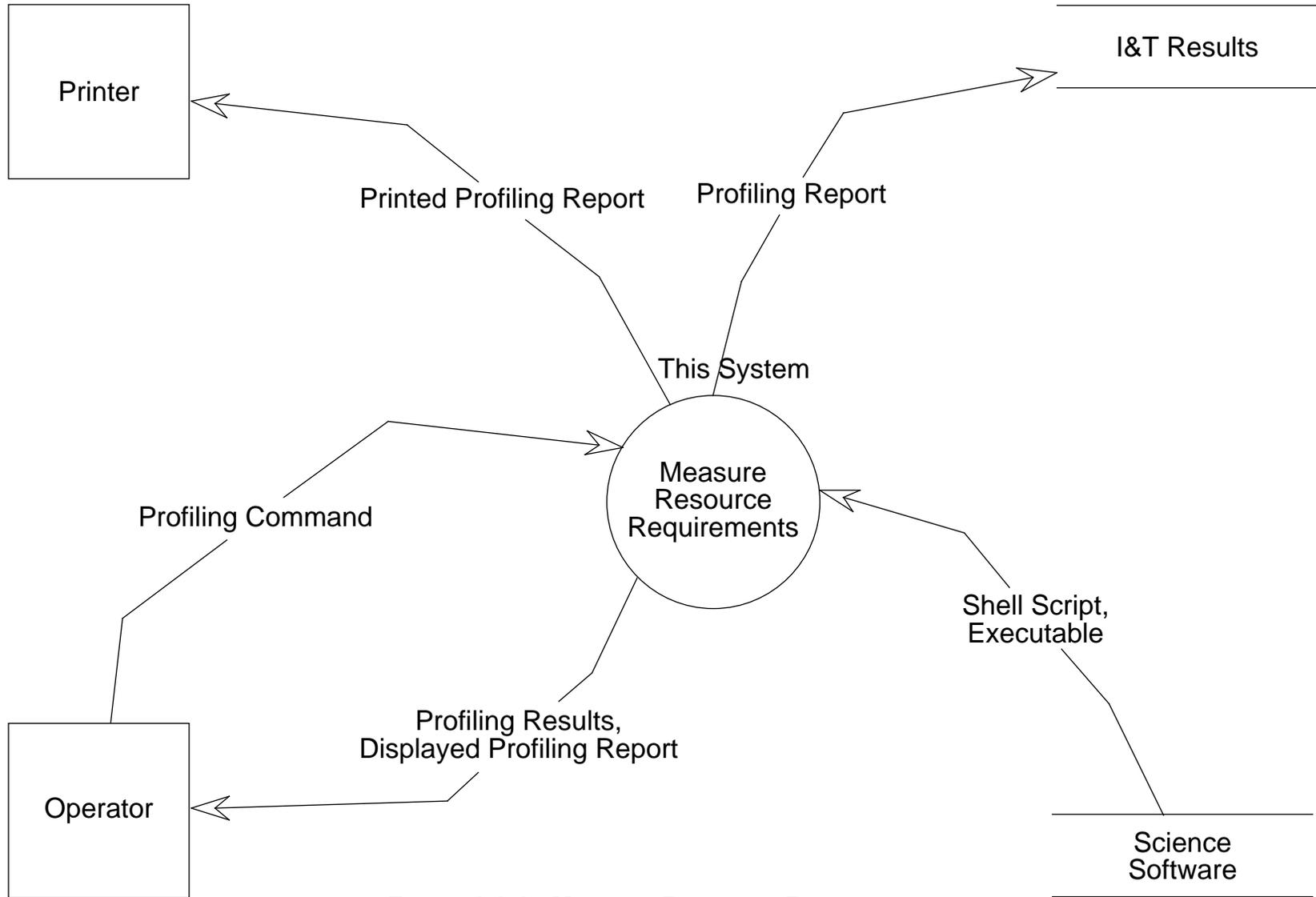


**Figure 6.6-5. Compare Files**

### **6.6.6 Measuring Resource Requirements**

The context diagram for the profiling tools is shown in Figure 6.6-6.

The operator issues a command (*Profiling Command*) to measure certain resource usage statistics, such as CPU time, memory usage, I/O accesses, disk space requirements, etc., for a specified process or procedure (*Shell Script, Executable*). The profiling results (*Profiling Results*) are displayed on the console. Reports may also be generated, and these may be displayed (*Displayed Profiling Report*), printed (*Printed Profiling Report*), or saved as a soft copy (*Profiling Report*).



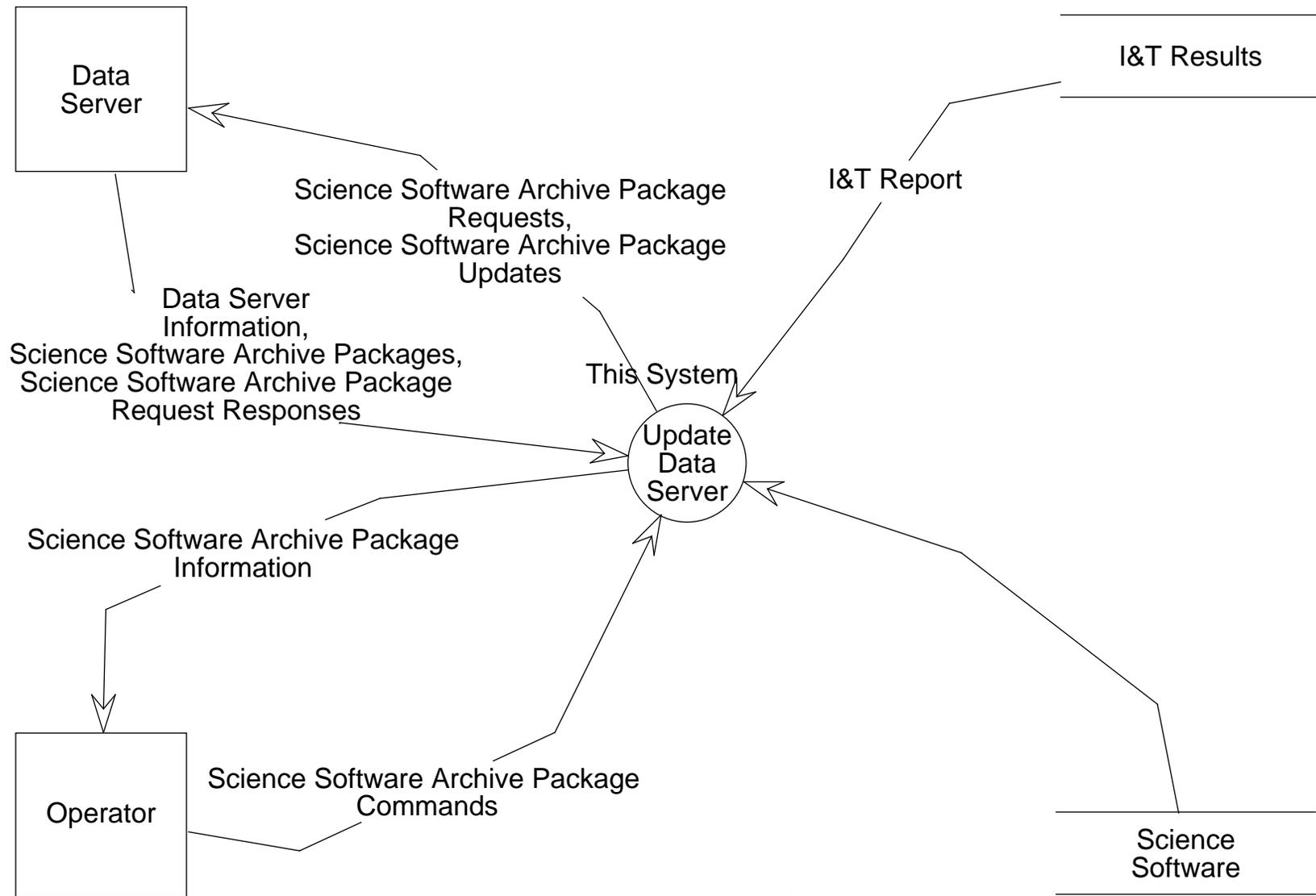
**Figure 6.6-6. Measure Resource Requirements**

### 6.6.7 Update Science Software Archive Package Tool

The context diagram for the archiving and updating of the Science Software Archive Packages on the Data Server is shown in Figure 6.6-7.

The purpose of this tool is to allow the user to add, delete, and modify Science Software Archive Packages at the Data Server once the executable software has passed acceptance testing. The set of files in a Science Software Archive Package may include not only the delivered files (*Science Software*) and their metadata, but also some of the reports (*I&T Report*) generated during integration and test. The operator issues various commands (*Science Software Archive Package Commands*), such as commands to view the current contents of an SSAP, to add new science software files to the SSAP, to update the metadata, and so on. In return, the results of operator commands, a list of SSAPs at the Data Server or the contents of an SSAP are displayed to the operator (*Science Software Archive Package Information*).

Requests (*Science Software Archive Package Request*) are sent to the Data Server to Acquire, Inspect, or Insert SSAPs (which includes the *Science Software Archive Package* to be Inserted); the Data Server responds to the requests with the requested information (*Data Server Information, Science Software Archive Package*), and a response (*Science Software Archive Package Request Response*).

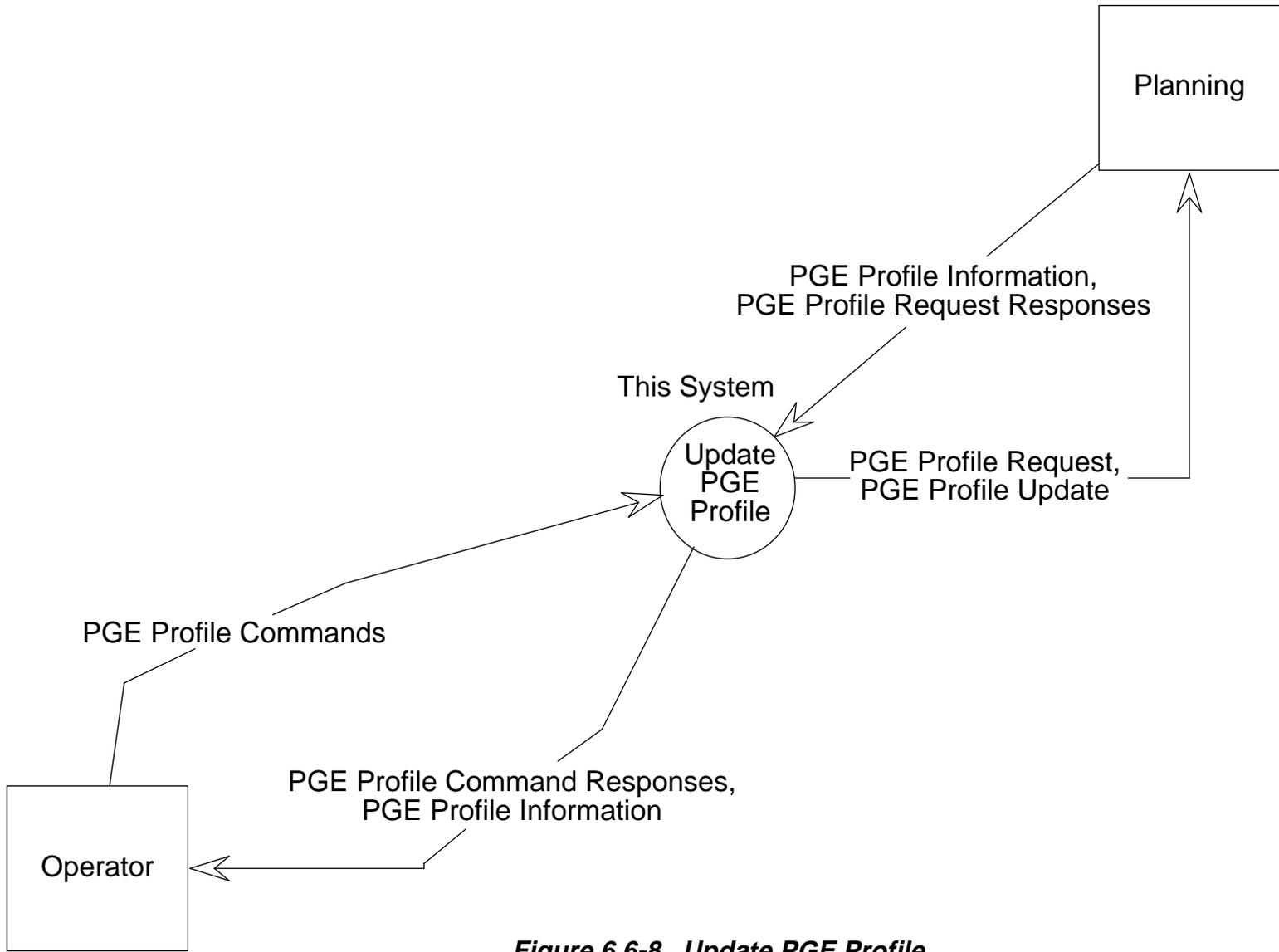


**Figure 6.6-7. Update SSAP**

### 6.6.8 Updating the PGE Profile

The context diagram for the tool for updating the PGE Profile is shown in Figure 6.6-8.

The operator issues commands (*PGE Profile Commands*) to add, delete, or view information from the PDPS database. The PDPS Database is a shared database used by Planning, Processing and Algorithm Integration and Test to manage their persistent data. These commands (*PGE Profile Commands*) are passed onto Planning (*PGE Profile Request*, *PGE Profile Update*) which performs the commands on the PDPS Database and returns responses (*PGE Profile Request Responses*) and information about the PGE Profile (*PGE Profile Information*). The information within the PGE Profile is passed back to the operator (*PGE Profile Information*) along with any command responses (*PGE Profile Command Responses*).

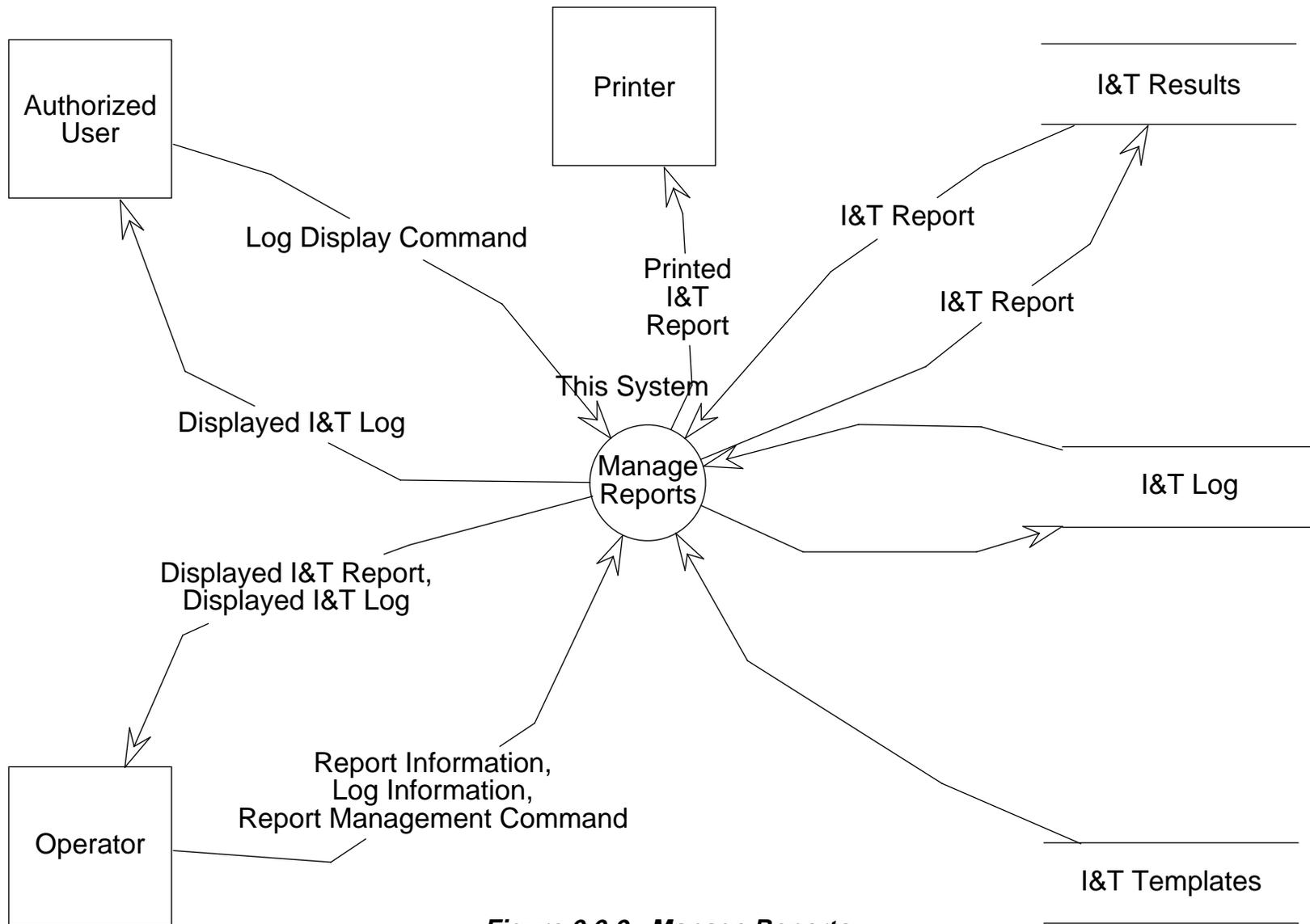


**Figure 6.6-8. Update PGE Profile**

### 6.6.9 Writing Reports and Maintaining Logs

The context diagram for tools to generate and maintain the integration and test reports and logs is shown in Figure 6.6-9.

The operator issues commands (*Report Management Command*) to create/modify specified reports (*I&T Report*) and logs (*I&T Log*), supplying the required information (*Report Information, Log Information*) and/or using information from other manually- or tool-generated reports (*I&T Report*), and using pre-generated report templates (*I&T Templates*). The operator may also display (*Displayed I&T Report, Displayed I&T Log*) or print (*Printed I&T Report*) any report or log, and any authorized user may display the log.

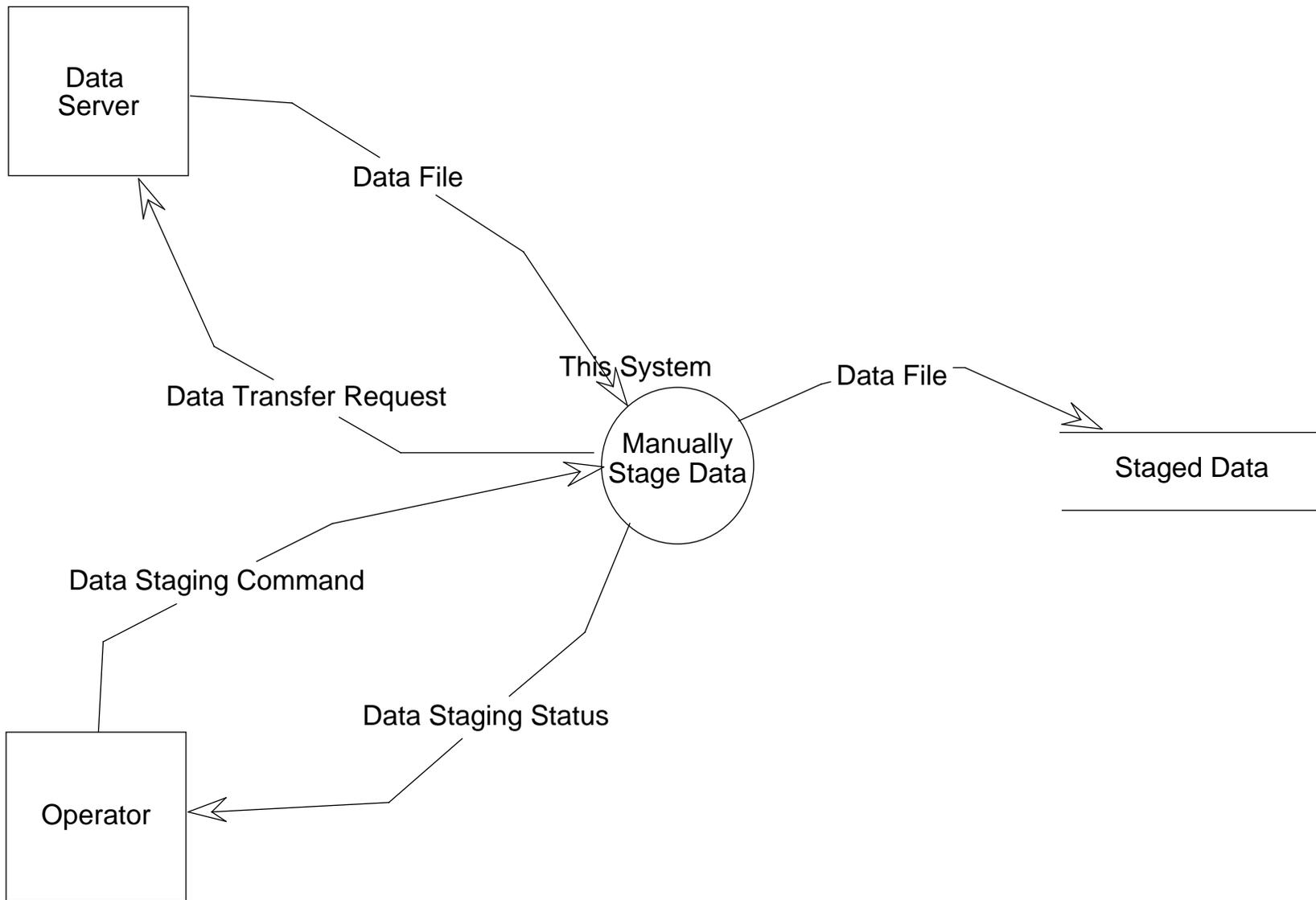


**Figure 6.6-9. Manage Reports**

### 6.6.10 Manually Staging Inputs

The context diagram for the tool for manually staging inputs is shown in Figure 6.6-10.

The operator issues commands (*Data Staging Command*) to manually stage a specified data file from the data server. The request (*Data Transfer Request*) is passed onto the data server, which returns the requested file (*Data File*), which is then placed in the staging area (*Staged Data*). Status (*Data Staging Status*) is displayed on the operator's console.

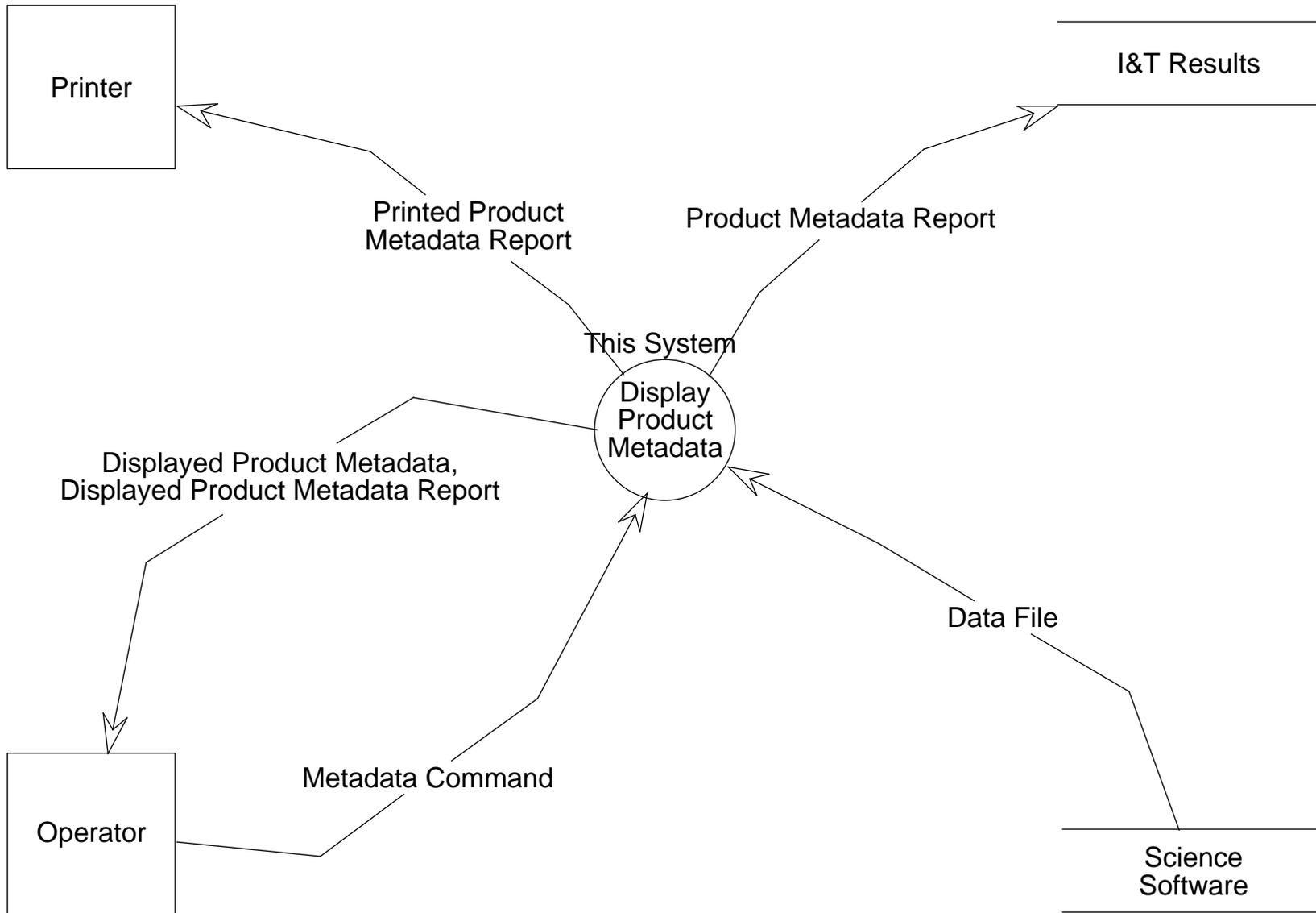


**Figure 6.6-10. Manually Stage Data**

### **6.6.11 Displaying Product Metadata**

The context diagram for the tool for displaying product metadata is shown in Figure 6.6-11.

The operator issues commands (*Metadata Command*) to display the product metadata (*Displayed Product Metadata*) for a specified data file (*Data File*). The metadata may also be extracted into a report, which may be displayed (*Displayed Product Metadata Report*), printed (*Printed Product Metadata Report*), or saved as a soft copy (*Product Metadata Report*).



**Figure 6.6-11. Display Product Metadata**

## 6.7 AITTL Operational Scenarios

The following three operational scenarios are intended to give an idea of how the science software integration and test process will be done, as well as illustrating where in the process the AITTL tools might be used by the SCF and DAAC I&T personnel. These scenarios were developed by the ECS Science Office and assume ECS Release B.

### 6.7.1 Engineering Version for AM-1

- 1) SCF sends 200 8-mm tapes containing test data files, expected test result files and associated metadata file. These tapes contain 11,000 files averaging 50 MB each.
- 2) DAAC Ingest - Distribution Technician mounts the delivered tapes and transfers these files to the Data Server/Archive.
- 3) DAAC Operations Supervisor has an Investigator Account opened for the SCF, an Investigator Directory created, and a CM storage pool allocated specifically for use in SSI&T.
- 4) The SCF transfers the science software (e.g., via ftp) to the Investigator Directory, and checks the source, includes, makefiles, scripts and libraries into the CM storage pool.
- 5) DAAC Resource Planner reserves 1 CPU of a Processing Server during day shift for use by SSI&T for the requested duration.
- 6) SCF remotely (e.g., via telnet) performs stand-alone testing of individual PGEs and PGE chains, employing the various AITTL tools in a “batch” mode (e.g., the data visualization tool is used to generate a graphics file, which is then transferred to the local site for display). Test data files are requested as needed from the Data Server/Archive.
- 7) When standalone testing is complete, SCF labels the “Delivery” elements in the SSI&T CM storage pool.
- 8) DAAC CM Administrator builds binary executables, checks out PGE scripts and submits these to Data Server (along with appropriate metadata).
- 9) DAAC Production Planner/Scheduler enters the test PGE information (e.g., PGE ID and Version No., resource profiles, inputs, activation rules) into the PDPS Data Base.
- 10) The SCF subscribes to the test PGE outputs.
- 11) DAAC System Tester requests test data from the Data Server and stages it to Ingest.
- 12) DAAC Production Planner/Scheduler schedules execution of test run through PDPS, processing is performed, and the PGE output and processing log files are automatically sent to the SCF with a data quality request notification.
- 13) SCF verifies PGE output.
- 14) The SCF checks the “Delivery” items out from the CM storage pool and transfers these elements (employing a DCE client) from the Investigator Directory to the SCF.
- 15) The DAAC performs cleanup work.

The CM Administrator deletes the SSI&T storage pool. The DAAC Production Planner removes the SSI&T PGEs from the planning Data Base and cancels any remaining Production Request(s) for those PGEs. The DAAC closes the Investigator Account and removes the Investigator Directory.

### **6.7.2 Launch-Ready Version for TRMM**

- 1) SCF requests that 11,000 previously delivered test data files be staged from deep archive to Data Server/Archive on-line storage.
- 2) The Data Server/Archive retrieves these files from deep archive.
- 3) DAAC Operations Supervisor has a User Account opened for the SCF, an Investigator Directory created, and a CM storage pool allocated specifically for use in SSI&T.
- 4)-14) Steps 4-14 of the “Engineering Version for AM-1” scenario are performed.
- 15) The DAAC CM Administrator uses the SSAP GUI tools to create a Science Software Archive Package entry in the Data Server and copies the “Delivery” elements from the SSI&T CM storage pool to the new Science Software Archive Package entry. Then the Administrator deletes the SSI&T CM storage pool.
- 16) The Production Planner/Scheduler enters Production Requests for product generation using the Launch-ready PGEs.

(The DAAC and SCF complete the operations readiness activities, and the new PGE is promoted to production status.)

### **6.7.3 Science Software Upgrade**

- 1) SCF requests that 24 previously delivered test data files be staged from deep archive to Data Server/Archive on-line storage.
- 2) The Data Server/Archive retrieves these files from deep archive.
- 3) DAAC Operations Supervisor has a User Account opened for the SCF, an Investigator Directory created, and a CM storage pool allocated specifically for use in SSI&T.
- 4)-14) Steps 4-14 of the “Engineering Version for AM-1” scenario are performed.
- 15) The DAAC CM Administrator uses the SSAP GUI tools to modify the original version of the Science Software Archive Package that contains the upgraded PGE in the Data Server and copies the “Delivery” elements from the SSI&T CM storage pool to the new SSAP entry. Then the Administrator deletes the SSI&T CM storage pool.
- 16) The Production Planner/Scheduler enters Production Requests for product generation using the new version of the Launch-ready PGE.

## **6.8 AITTL Structure**

Table 6-1 provides a list of the Computer Software Components in the AITTL CSCI and Table 6-2 provides a mapping of objects to CSCs.

Note that all software is callable from the UNIX command line; alternatively, it may be called from the AIT Manager GUI. Command line versions are shown in brackets [], where they differ from the GUI versions.

In implementing custom code, some SDP Toolkit functions are reused, primarily to track AIT configuration files and manage temporary files.

**Table 6-1. AITTL Computer Software Components**

<b>CSC</b>	<b>Description</b>	<b>Implementation</b>
Documentation Viewing Tools	Tools for displaying and/or printing the science software documentation	SoftWindows/MS Office Ghostview
Standards Checkers	Tools for checking if science software follows prescribed coding standards.	Native compilers] FORCHECK
Code Analysis Tools	Tools for checking for code memory leaks, etc.	CASEVision SPARCWorks
Data Visualization Tools	Diagnostic tools which display input, output, and intermediate data files as data dumps, plots, and/or images.	IDL
ECS HDF Visualization Tools	Provides the capability to view any ECS-HDF formatted files.	EOSView
HDF File Comparison Utility	Tool for finding differences between two HDF files,	DpAtMgrCheckHdfFile
Binary File Comparison Environment	Tool for assisting DAAC user in writing custom code to find differences between two binary files	DpAtMgrBinaryFileEnvironmentGui
Profiling Tools	Tools for measuring the resource requirements of the science software	CASEVision
Science Software Archive Package GUIs	GUIs for view and altering Science Software Archive Packages	DpAtSSAPGuiNB DpAtEditSSAPFileListGuiNB DpAtEditSSAPMetaDataGuiNB DpAtAccessNB
Acquire and Insert GUI	GUI used for staging (Acquiring) or destaging (Inserting) data.	DpPrAITManualIF
PGE Registration GUIs	GUIs for registering a PGE with PDPS	DpAtPgeRegistrationGui DpAtPgeUserParametersGui DpAtPgeDataTypesGui DpAtPgeActivationRuleB
Report Generation Tools	Tools for writing miscellaneous reports and for maintaining the integration and test log	SoftWindows/MS Office DpAtMgrCom
SDP Toolkit-related Tools	Tool to check Process Control File format; tool to check that no prohibited functions are used	DpAtMgrCheckPcfGui [pccheck.sh] DpAtMgrCheckProhibFunc [DpAtMgrCheckProhibFuncCom]
Product Metadata Display Tool	Tool for displaying the product metadata	DpAtMgrCheckHdfFile

**Table 6-2. Mapping of objects to CSCs**

<b>Object</b>	<b>GUI</b>
DpAtPGERegistrationFile	PGE Registration GUIs
DpAtPgeActivationRuleB	PGE Registration GUIs
DpAtPgeDataTypes	PGE Registration GUIs
DpAtPgeRegistationGui	PGE Registration GUIs
DpAtPgeUserParameters	PGE Registration GUIs
PIAlternate	PGE Registration GUIs
PICluster	PGE Registration GUIs
PIDataScheduled	PGE Registration GUIs
PIDataTypeB	PGE Registration GUIs
PIDataTypeReqB	PGE Registration GUIs
PIOrbitModelNB	PGE Registration GUIs
PIOrbitScheduledNB	PGE Registration GUIs
PIOutputYield	PGE Registration GUIs
PIPGE	PGE Registration GUIs
PIPGEProfile	PGE Registration GUIs
PIPerformance	PGE Registration GUIs
PIResourceRequirement	PGE Registration GUIs
PITile	PGE Registration GUIs
PITileScheduledNB	PGE Registration GUIs
PITimeScheduled	PGE Registration GUIs
PIUserParameters	PGE Registration GUIs
DpAtAccessNB	Science Software Archive Package GUIs
DpAtEditSSAPFileListGuiNB	Science Software Archive Package GUIs
DpAtEditSSAPMetaDataGuiNB	Science Software Archive Package GUIs
DpAtSSAPFile	Science Software Archive Package GUIs
DpAtSSAPGuiNB	Science Software Archive Package GUIs
DsCICommand	Science Software Archive Package GUIs
DsCIESDTReferenceCollector	Science Software Archive Package GUIs
DsCIRequest	Science Software Archive Package GUIs
GIPParameter	Science Software Archive Package GUIs
GIPParameterList	Science Software Archive Package GUIs
DpPrAITManuallF	Acquire and Insert GUI
DsCICommand	Acquire and Insert GUI
DsCIESDTReferenceCollector	Acquire and Insert GUI
DsCIRequest	Acquire and Insert GUI

### 6.8.1 Documentation Viewing Tools

This CSC contains the tools that the integration and test personnel will use to view the science software documentation. The tools only need to be able to display a document on a console and print the document—there is no editing capability required. The list of formats that will be accepted is given in 304-CD-002-001, Science and Data Processing Segment (SDPS) Requirements Specification for the ECS Project (see requirements S-DPS-40100 and 40110).

The Microsoft Office automation tools, in conjunction with the SoftWindows Windows emulator for Sun, is used here.

## **6.8.2 Standards Checkers**

This CSC contains the standards checking tools. Native language compilers satisfy the requirements here, except in the case of FORTRAN 77. For that language, a COTS standards checker is used, because most if not all compilers support a near-standard subset of ANSI FORTRAN 77; it is expected that essentially all FORTRAN 77 science software will contain these extensions to the ANSI standard.

## **6.8.3 Code Analysis Tools**

These tools are for enabling DAAC personnel to determine the causes of such problems as memory leaks. They include the powerful analysis environments SPARCWorks (on the Sun) and CASEVision (on the SGI).

## **6.8.4 Data Visualization Tools**

This CSC contains the data visualization tool IDL, required by the integration and test personnel to examine input, output, and intermediate data files for diagnostic purposes (but not for QA).

## **6.8.5 ECS HDF Visualization Tools**

This CSC is the ECS-developed EOSView tool for viewing ECS HDF files. The tool is reused from the WKBCH CSCI.

## **6.8.6 HDF File Comparison Utility**

This CSC contains the HDF file comparison utility. It is implemented by a custom IDL program, which includes a GUI front end. Difference data may optionally be displayed as text, as a line graph with tolerances, or printed.

The first time a given HDF standard product is compared, the DAAC user must manually input data tolerances that s/he reads by eye from a text file delivered with the science software.

## **6.8.7 Binary File Comparison Environment**

This CSC contains the binary file comparison environment. This is implemented by a GUI from which the user can choose example code and function utilities to cut and paste for making a custom file differencing tool, tailored to the particular binary file format. This format is also available from the GUI, as delivered in a text file along with the science software.

## **6.8.8 Profiling Tools**

This CSC contains the profiling (i.e., resource requirement measurement) tools. These tools must satisfy any of the AITTL profiling requirements that are not satisfied by the development environments or operating systems supplied by the AITHW CI. CASEVision covers this.

## **6.8.9 Science Software Archive GUIs**

This CSC contains the GUIs used to view and alter Science Software Archive Packages at the Data Server. The functionality is supplied by classes DpAtSSAPGuiNB, DpAtEditSSAPFileListGuiNB, DpAtEditSSAPMetaDataGuiNB, and DpAtAccessNB.

### **6.8.10 Acquire and Insert GUI**

This CSC contains the GUI/tool that provides an interface to the Data Server for manually staging (Acquiring) and destaging (Inserting) of data files. This functionality is supplied by DpPrManualIF.

### **6.8.11 PGE Registration GUIs**

This CSC contains the GUIs used to register a PGE in the PDPS database. This functionality is supplied by DpAtPgeRegistrationGui, DpAtPgeUserParametersGui, DpAtPgeDataTypesGui, DpAtPgeActivationRuleB.

### **6.8.12 Report Generation Tools**

This CSC contains the tools to write and maintain the integration and test reports and logs. All but the log requirements are satisfied by SoftWindows/MS Office; the AIT Manager covers the need for a log.

### **6.8.13 SDP Toolkit-related Tools**

These tools (a) check the format of a delivered Process Control File and (b) determine whether any prohibited functions are used, which might interfere with the processing system. Modules DpAtMgrCheckPcfGui (which is a wrapper on an existing SDP Toolkit utility) and DpAtMgrCheckProhibFuncGui are used, respectively.

### **6.8.14 Product Metadata Display Tool**

This CSC contains the tool for displaying product metadata. It is implemented by DpAtMgrCheckHdfFile, the same tool as in "HDF File Comparison Utility" above.

## **6.9 CSCI Management and Operation**

### **6.9.1 System Management Strategy**

This section discuss the management and operation of the AITTTL CSCI. It addresses how the CI is managed at the local level and supports system level management and operations.

Three key concepts define the management and operations of this CSCI. These are that:

- a. The AIT process is essentially standalone relative to the rest of the ECS system
- b. AIT is operations and operational procedure driven.
- c. The AITTTL CSCI must support the flexibility of operations required of an I&T activity.

The following paragraphs discuss these features further, indicating how the AITTTL CSCI supports this view of management and operations.

#### **Independent Operations**

From the point of view of system management and operations, the AITTTL operates in essentially a standalone mode, apart from the rest of the ECS. Its purpose is to support the integration and test of Science Software with the ECS system at the DAACs. This is a non-operational CSCI in the sense that it is not involved with the handling or processing of science telemetry from EOS supported spacecraft or instruments. During the operational phases, hardware elements on which

the AITTL will run during integration and test (I&T) will be specially configured so that they do not interfere with operations, and operations does not interfere with the test process.

The AITTL does not participate with the system management services provided through MSS for operational enterprise management, such as fault management and startup & shutdown. Note however, that the hardware platforms on which the AITTL tool set is used during I&T will be monitored via MSS agents for hardware faults. The tool set makes use of MSS provided configuration management tools to place the science software under configuration management during the I&T process for eventual migration into operational configuration control databases. The operational procedures AITTL call for the use of ECS standard 'trouble ticketing' utilities for recording of problems identified during I&T. Other system services, such as ingest and data server access, will be used to establish the environment for the I&T process. But the I&T process and the AITTL CSCI will be decoupled from the operational ECS system.

### **Operations Procedure Driven**

A key concept for the operation involving AITTL is that the AIT process, as with almost any integration and test activity, is a people-intensive activity. It is driven by the operational procedures established by the DAAC-AIT team in consultation with the science software developers. The management of the AITTL is dictated by the management of AIT operations procedures. The tool set supports this view in that it is a collection of tools that is called upon, one at a time, to address one of the steps in the sequence of activities in the procedure. Evolution of the I&T operations procedures (as the result of increased understanding of the process and the software) is supported because there is no underlying assumption within the tool set implementation with regard to the details of the I&T operations procedures.

## **Flexibility of Operations**

A critical feature of the AITTL CSCI is that it must exhibit great flexibility to support the dynamic environment that is a part of the I&T process. The AIT process, particularly during the TRMM Release period, will be greatly constrained in the time allotment. The personnel (operations, ECS, and science software) involved in the process must be able to adapt quickly to address unexpected situations as they arise during I&T. The tool set is flexible enough to support the many variations that can be expected to occur during the many science software integration and test events. Again, because the tool set does not assume a particular operations procedure, the tools can be readily adapted to variations in the procedures, from one instrument team to another. The tool set can support adaptation of the procedure 'on-the-fly' as problems are detected and alternate approaches to the I&T process are investigated. This is because the AITTL CSCI is a collection of tools that support the I&T operations procedure—as the operations procedure is updated or modified, the tool set support can be redirected.

### **6.9.2 Operator Interfaces**

This subsection describes the operator user interfaces provided by the AITTL CSCI to DAAC operations personnel. A general description of the framework and methodology employed for the development of these interfaces can be found in Section 4.5. of the Detailed Design Overview (305-CD-001-001). This subsection augments that information with additional design information which is specific to the Algorithm Integration & Test CSCI.

The operator user interfaces for the algorithm integration & test environment are custom and COTS provided interfaces. This custom graphical interface will be created with the aid of the Integrated Computer Solutions' Builder Xcessory. Builder Xcessory enables the developer to manage Motif graphical user interface projects by providing a WYSIWYG, drag and drop, visual development environment. Once an interface is constructed, Builder Xcessory will generate C++ code which represents the GUI and encapsulates the C-based Motif Widget set. The generated C++ code can then be combined with other AITTL CSCI specific code.

#### **6.9.2.1 Algorithm Integration & Test CSCI User Interfaces**

This section is intended to describe the data that may be displayed for the operations of the AITTL CSCI's applications. The exact definition of the GUIs will be decided by requesting user suggestions and through demonstrating prototypes.

##### **AI&T Manager GUI**

The AI&T Manager GUI is an interface which provides the operations staff with the capability to perform various AI&T activities which are defined as data visualization, updating quality assurance metadata, and subscribing to data. The AI&T Manager GUI will be constructed with custom code which interfaces to the PDPS Database for storage of PGE Profile and other AI&T persistent data.

As part of the GUI, utilities will be provided to initiate data visualization tools, such as EOSVIEW, which will be used to visualize a science data product. Other tools needed to perform AI&T duties, including standards checkers, memory analyzers, and office automation tools can also be invoked from this tool. Also, provided is an interface to manually initiate the staging of data to or destaging of data from the Science Data Server CSCI.

## **AI&T Science Software Archive GUIs**

The AI&T Science Software Archive GUIs provide the operations staff with an interface to the capability to perform alterations on the Science Software Archive Packages stored at the Data Server. It is started from the UTILITY menu of the AI&T Manager GUI.

## **AI&T PGE Registration GUIs**

The AI&T PGE Registration GUIs provide the operations staff with an interface to the capability to alter/view PGE Profiles in the PDPS database. It is started from the UTILITY menu of the AI&T Manager GUI.

### **6.9.3 Reports**

A variety of ad-hoc and canned reports will be available to the DAAC operations staff to assist in the monitoring of the activities associated with the Algorithm Integration & Test CSCI. These reports are readily accessible given that the Algorithm Integration & Test CSCI persistent data is maintained in the PDPS Database, a SYBASE RDBMS. Also, ECS application management information is maintained in the MSS database, which is used to log system events. The canned reports will include the following:

- a. PGE Profile Reports—these reports will be used to catalog the resource profile information associated with a PGE. This information, which includes generation size of PGE Output data, CPU Wall Clock Time Used, CPU actual time Used, I/O Operations, etc., is captured over a series of PGE executions. A profile will be captured for each type of machine, i.e., Sun, SGI, etc., for which the PGE is to execute. Statistics will be collected to establish standard deviations, variances, and averages of resource profile values. These reports will be used to collate this information for a PGE, for a type of resource, or for a given group of PGEs used to fulfill a Production Request.
- b. I & T Activity Report—these reports will capture information about the activities which have occurred and activities which are occurring in the Algorithm Integration and Test environment.
- c. PGE Profile Update Report—these reports will capture information to track the updates which have occurred in the PDPS Database to PGE Profiles.
- d. PGE I&T Reports—these reports capture information on PGEs as they progress through the AI&T process. These reports will be used to trouble shoot problems and will provide tracking and trend analysis guidance.
  1. Code Analysis Report
  2. Standards Checker Report
  3. File Comparison Reports
  4. AI & T Discrepancy Reports
  5. Inspection Reports
  6. Integration Reports
  7. Acceptance Reports

- e. Algorithm Integration & Test Management Reports—These reports will provide the operations staff information on Algorithm Integration and Test application software events which have occurred. This information will be available from the MSS database.

Other ad-hoc reports can be defined to assist the Algorithm Integration & Test Operations staff in performing their activities. The PDPS Database is the repository used to maintain information on Production Requests and associated Data Processing Requests, Data Subscriptions, PGE Profiles, etc. These reports can be used to track modifications and provide historical information on these data objects. Because of the use of a consistent RDBMS throughout ECS, the sharing of information between different databases is simplified and will allow for consistent definitions for any number of reports.