Enhancing Reusability of Java HDF

April 21, 2003,

HDF Java Team
Robert E. McGrath, Peter Cao, Mike Folk
mcgrath@ncsa.uiuc.edu

1. Overview

This note describes proposed revisions and redesign of the HDFView Java tool, to be done this summer. The goal of this work is to make the HDF Java code more reusable, and make it easier for users to customize the HDFView tool by replacing or adding modules.

The fundamental approach is to re-architect the HDFView code to have several internal interfaces that enable users to write and use alternative implementations of our standard modules.

The replaceable modules will be:

- File I/O (already implemented)
- Image view
- Table view (a spreadsheet-like layout)
- Text view
- Metadata (metadata and attributes) view
- Tree view
- Palette view

The desired features are:

- Abstract interfaces along with a default implementation
- A mechanism for selecting which implementation to invoke (when > 1). In the case of Image, Table, and Metadata views, this must be selectable per object.
- Extensive documentation, examples, and user's guide of how to make such a module.

2. Example: Abstract I/O Layer

The current implementation has an abstract I/O layer that illustrates the proposed approach. This section explains the existing I/O layer.

The abstract I/O layer defines basic abstract classes such Group, Dataset, Datatype and Metadata. These abstract classes contain APIs for data access such as read data from a file or write data to a file. Application software should build on this abstract layer so that the application will not depend on the implementation of the I/O objects. The current I/O layer, *ncsa.hdf.object*, is an example of the approach.

The following diagram explains the structure of the HDF object package and how to use it. Package *ncsa.hdf.object* contains the common object/interface. This package only defines interfaces or abstract classes for accessing HDF files. The implementation will rely on the HDF4/HDF5 object package. Application classes should only include *ncsa.hdf.object*.

The *ncsa.hdf.object.h4* is an implementation of the HDF4 objects. It implements the interfaces and abstract classes defined at *ncsa.hdf.object* package. The HDF4 object package requires the HDF4 Java Native Interface (*ncsa.hdf.hdflib*). Application should not include this package so that it compiles and runs without the HDF4 library.

The *ncsa.hdf.object.h5* is an implementation of the HDF5 objects. It implements the interfaces and abstract classes defined at *ncsa.hdf.object* package. The HDF5 object package requires the HDF5 Java Native Interface (*ncsa.hdf.hdf5lib*). Application should not include this package so that it compiles and runs without the HDF5 library.

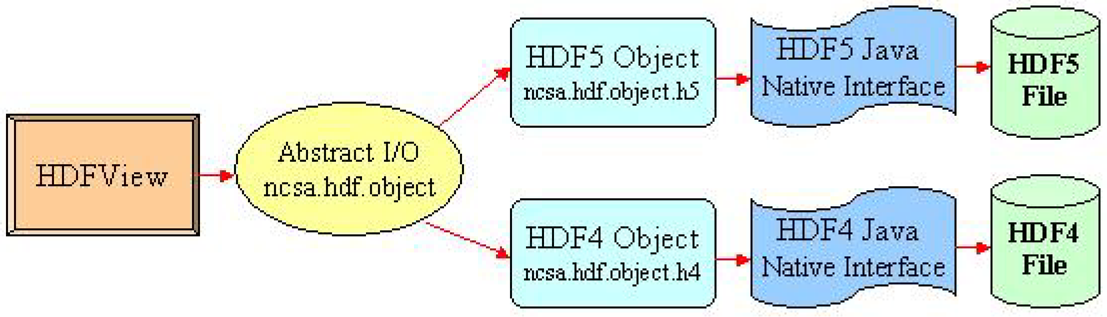

Figure 1

When the application opens an existing file, the file is examined and the HDF4 or HDF5 package is used, as needed. If the required library is not configured or the file is not HDF, then the open fails. After the file is opened, the application uses only the methods of the *ncsa.hdf.object* package, which are implemented with the correct format-specific implementation.

When the application creates a new file, it specifies whether to create HDF4 or HDF5. This option is used to select which implementation of the *ncsa.hdf.object* to use.

Some operations of the *ncsa.hdf.object* are not fully supported in each implementaton (e.g., object deletion). In this case, the operation will fail if the format specific code cannot do it.

3. Abstract GUI Interfaces

We propose to extend the approach used in the abstract I/O layer to parts of the GUI interface. The purpose of the abstract GUI interface is to make the GUI components modular and easy to reuse the code. Users can write their own GUI components implementing from the abstract GUI interface to replace the default HDFView GUI components so that they can display the data in their own way. Also, because of the modular design of the GUI components, users can easily adopt and reuse the HDFView source code.

The abstract GUI components include the TreeView, ImageView, TableView, TextView, and MetadataView (for metadata and attributes). (Figure 2) Other GUI components may be added at user request. Default implementing classes of the abstract GUI modules are provided. The TreeView defines the interfaces that manipulate the file structure such as add/delete data object. The ImageView, TableView, and TextView define APIs that read/write data information and data object. MetadataView defines interfaces to read/write metadata and attributes.

The following is an example of the ImageView interface.

```
public interface ImageView
{
    /**
       Returns the data object of the image.
     */
    public abstract Object getDataObject();
     /* … other methods TBD … */
}
```

This interface would be called, for example, by the TreeView to determine what kind of object it is (e.g., to display an appropriate icon in the tree).

Each implementation would be responsible for processing and displaying the data, e.g., an implementation of ‘ImageView’ would read the data and display it in an image window. (Details of how this will be displayed within the HDFView are TBD.)

In addition to the display functions, the implementation will be responsible for all the control buttons and functions. These will be moved from the main menu bar to the respective GUI component. E.g., in V1.2 of HDFView there is an ‘Image’ menu on the main menu bar. This will be moved to implementations of the ImageView interface. (Standard Java provides the interfaces and frameworks to quickly create menus and menu bars, as needed.)

There may be several user implemented modules as well as the default module. Multiple implementations may be configured and used in the tool. The default module may be included or not, and any number of user modules may be included.

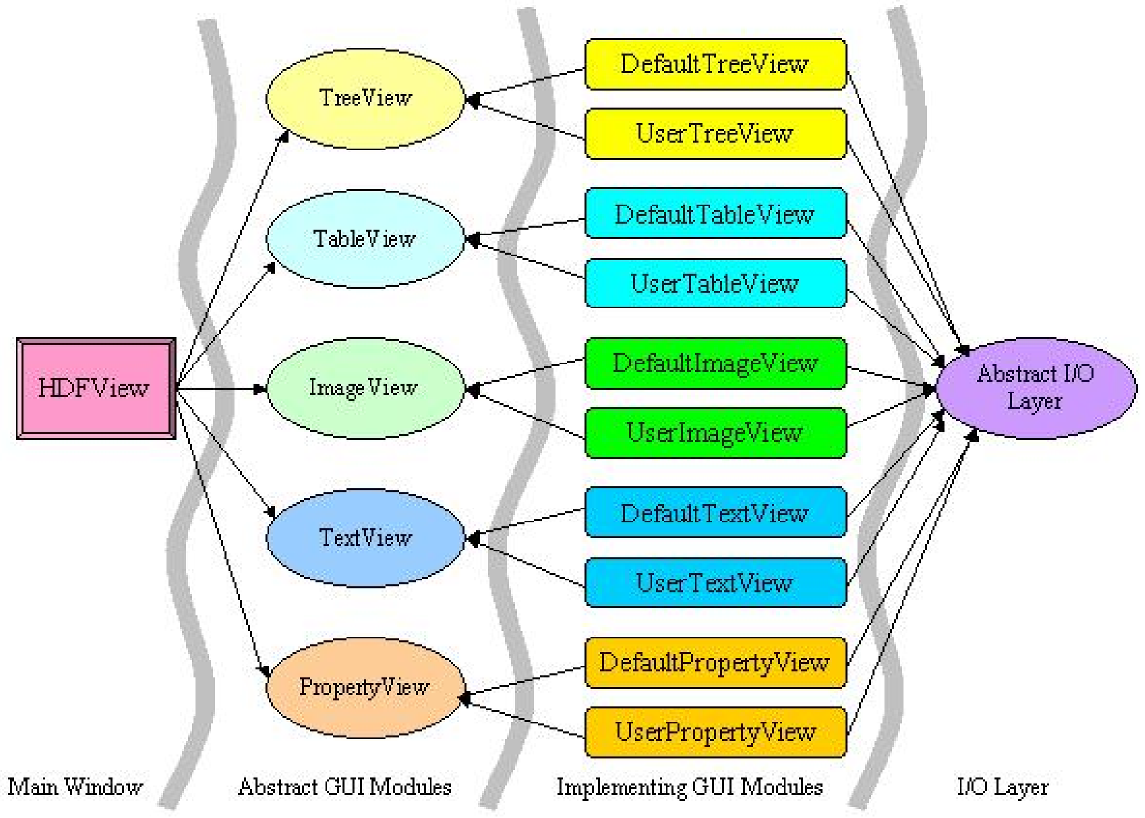

Figure 2

Registration and Selection of Modules

User modules will need to be registered with the HDFView tool. The precise mechanism is TBD, but it will probably be done by copying a JAR file to a directory and updating a configuration file. The tool will look at the configuration file and load the module(s) from the JAR file.

When there is more than one implementation of an interface, there should be a mechanism to select which module to call for each object. I.e., when an image is opened, the ‘Open As’ dialog should show the alternative image viewers that are configured. (The configuration file probably needs an information string to identify the module.)

Alternatively, customized modules may automatically select the view based on application knowledge. E.g., a customized tree viewer may know which images should be displayed with the default and which should be handled by a custom image view, and launch the correct viewer without asking.

This mechanism is not completely worked out at this time.

Customization of Attributes

In the current design, the 'MetadataView' is a single module that displays all the metadata and attributes for an object. This is a complex and multi-function module, so it is not totally clear how to modularlize it.

A simple approach is to make a single abstract 'MetadataView' interface, which users can implement. This is shown in Figure 2: the MetadataView is treated analogously to the other modules.

However, in order to customize one part of the metadata or a single attribute will require providing a substitute module for all the information in this window. It seems likely that the 'metadata' page (dimensions, datatype, compression, etc.) should be a separate module from the 'attributes' page.

One idea that might be considered is to have an abstract 'AttributeViewer', which is called for each attribute. The default implementation returns the strings needed to fill in the table in the current display. An alternative implementation could return whatever strings it wants, or do something else entirely. In the latter case, the 'AttributeViewer' will return a value to indicate that the attribute should not be displayed in the table. This approach will let people handle specific attributes, e.g., EOS Metadata. But it is not clear how easy this would be to implement, and it is not clear how to associate different handlers with specific attributes.

Another desirable feature would be the ability to present all attributes in a single page. Meeting all the desired goals for attributes and metadata will require some study.

4. Summary of Project

This project requires the rearchitecting of the HDFView tool. At the end of the project, the HDFView tool will have the same function and appearance, but will be implemented in replaceable modules.

This project has the following important deliverables:

New release of HDFView with the same features as the current tool, built with the new architecture.
Abstract interfaces along with a default implementation of each interface.
A mechanism, such as a configuration file, for adding alternative modules to the viewer.
A mechanism for selecting which implementation to invoke (when > 1). In the case of Image, Table, and Metadata view, this must be selectable per object.
Extensive documentation, examples, and developers' guide of how to make such a module.

The design will implement interfaces for the following modules (subject to revision during design studies):

File I/O (already implemented)
Image view
Table view
Text view
Metadata view (metadata and attributes)
Tree view
Palette view

The documentation will include:

Complete specification of the interfaces and requirements for implementation modules
Documentation of the default implementation sufficient to be an example for implementers.
Developer's guide with examples of how to create and deploy modules.

5. Estimated effort

This project has not been scoped in detail. An initial estimate would be that this is 2-4 months work. This work could probably begin in May 2003.