Astronomical Data Analysis Software and Systems VIII
ASP Conference Series, Vol. 172, 1999
D. M. Mehringer, R. L. Plante, and D. A. Roberts, eds.

Design Study on Integration of StarView II and JIPA Using Advanced Java Mechanisms

Markus Dolensky

Space Telescope – European Coordinating Facility, Karl-Schwarzschild-Str. 2, D-85748 Garching, Germany

Bruce Mayhew, Bridget Kennedy

Space Telescope Science Institute, 3700 San Martin Drive, Baltimore, MD 21218

Abstract. This case study examines the use of the latest software techniques for merging independent Java applets and applications. STScI's new Java interface to the archive, StarView II, will make use of the ST-ECF stand-alone FITS image viewer JIPA. We discuss how state-of-the-art software techniques like Remote Method Invocation (RMI), multi-threading, and the Reflection API influence the design of a common interface between StarView II and JIPA. Furthermore, it is demonstrated how the JDBC interface will provide the two applications with concurrent access to the archive.

1. Merging Java Components – StarView II & JIPA

STScI's prototype of the new Java interface to the archive, Starview II (SV II) makes use of ST-ECF's[1] stand-alone FITS image viewer JIPA[2]. The interface between these independent Java components is based on two design patterns:

- Observer Pattern (Fig. 1)
- Design Inheritance (Fig. 2)

2. Observer Pattern

The intent of the observer pattern is a way to define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and update automatically (see, for example, `http://www.cs.wustl.edu/~schmidt/cs242/patterns.html`).

[1] `http://ecf.hq.eso.org/`

[2] `http://archive.eso.org/java/jipa/`

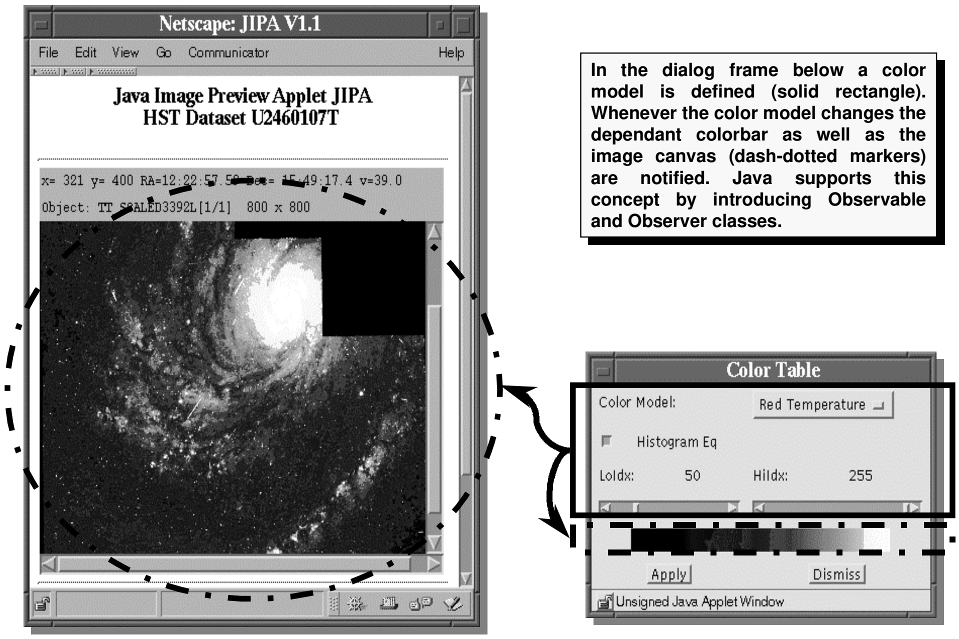

Figure 1. Snapshot of JIPA using the Observer Pattern

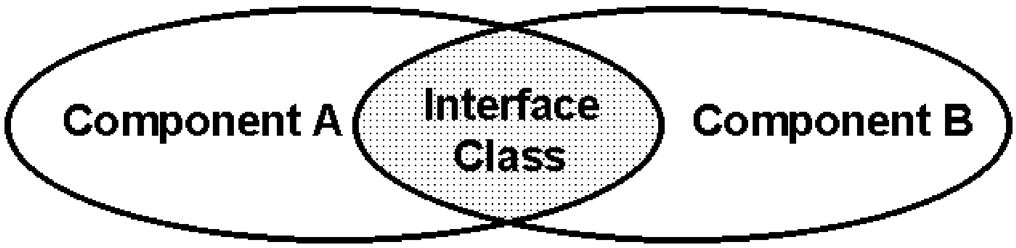

Figure 2. Design Inheritance

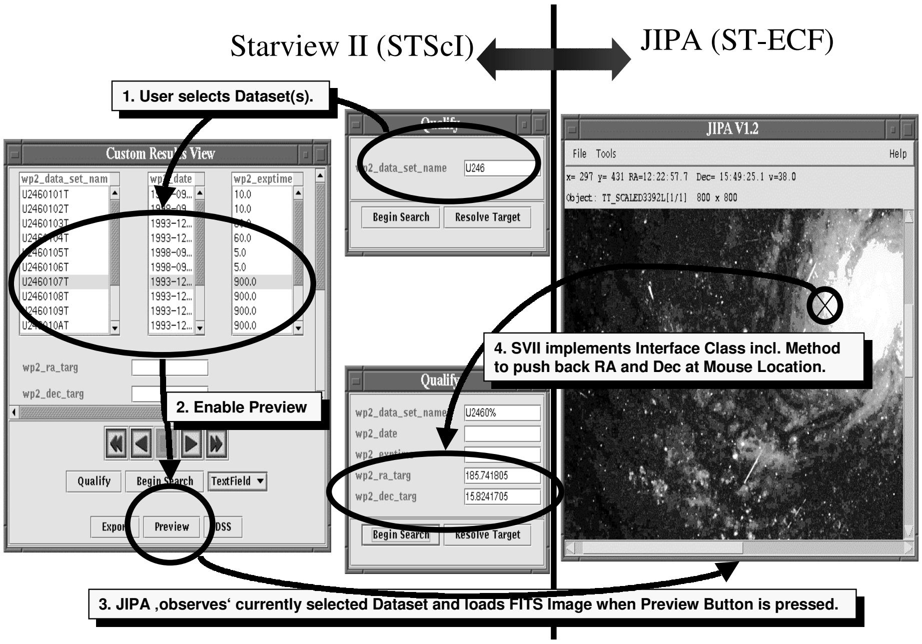

Figure 3. Screenshot of StarView II & JIPA and how they interact

3. Design Inheritance

3.1. What Are Java Interfaces?

An interface is a collection of method definitions (like C function prototypes) and constant values (Campione & Walrath 1998)[3].

3.2. Implementing and Using Interfaces

A class implementing an interface provides a method implementation for all the prototypes declared within the interface and its superinterfaces. The interface is a new reference data type which can be used as any other type name in variable declarations and method parameters.

3.3. Why Interfaces?

The interface scheme provides an elegant way to separate design and implementation. It is possible to develop (compile) two Java components independently – they just share the interface source file containing the constants and method declarations (Fig. 2).

[3]http://java.sun.com/docs/books/tutorial/java/more/interfaces.html

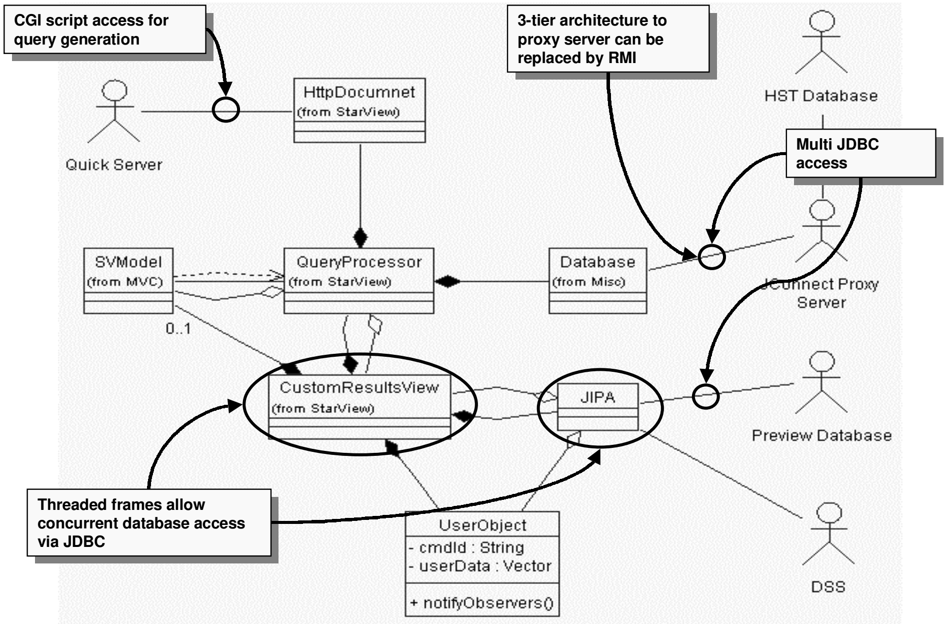

Figure 4. Server Access Architecture of StarView II

4. Server Access Architecture

The archive browser StarView II has various interfaces (Fig. 4). Apart from the user interface, there is the Quick Server which translates queries into SQL statements. Furthermore, there is the JDBC interface to the actual database, and finally there are two more interfaces which communicate with other Java components as described above. Again, these two are the observer/observable pattern (implemented as class UserObject) and design inheritance. As illustrated, JIPA has its own means to access images from the preview DB and DSS catalogue.

Currently, a mixture of CGI (Common Gateway Interface) and JDBC (Java Database Connectivity) is used for network connections. In the future, the three-tier JDBC architecture to a proxy server, which is imposed by Java security restrictions, might be replaced by RMI (Remote Method Invocation).

References

Campione, M. & Walrath, K. 1998, The Java Tutorial (2nd ed.), (Reading: Addison-Wesley), 145