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

StarView II: A Customizable Query Generation Tool

Bridget Kennedy, Bruce Mayhew

Space Telescope Science Institute

Abstract. StarView is the user interface to the Hubble Space Telescope (HST) Data Archive. It supports queries to the HST catalog, submission of retrieval requests, data preview, and other archive-related functions. This presentation looks at the StarView II effort to make a flexible, adaptive framework for browsing astronomical catalogs. We discuss user interaction with the customizable form, describe the use of Java Foundation Classes in implementing the Model View Controller design pattern, and present design features that allow simple, robust communication with third-party components added at run-time.

1. Introduction

StarView II addresses two important concerns within the astronomical community: the desire for a flexible catalog browsing tool and the need for an adaptive framework capable of seamless interaction with current and future functional components.

2. Flexible Browsing Environment

Along with standard query forms, StarView II supplies an interactive custom query form building tool – the Customizer. As shown in Fig. 1, the Customizer contains a keyword search panel and hierarchical displays of the data structure. With these catalog data browsing components, the user builds custom query forms through simple point-and-click interaction. All forms maintain their own state information, permitting asynchronous execution of multiple searches. Each query form, along with its specific formatting and state information, can be saved and retrieved using Java object serialization.

StarView II forms implement the Model View Controller (MVC) design pattern through encapsulated data modeling and selectable data views that interact via a controller. Clear division of functionality, and strategic use of interface classes, result in a system that readily accommodates varying data sources and view components.

2.1. Data Models

A simple two-dimensional vector was chosen for the structure of the base data model. Application-specific knowledge of the data structure is contained in a

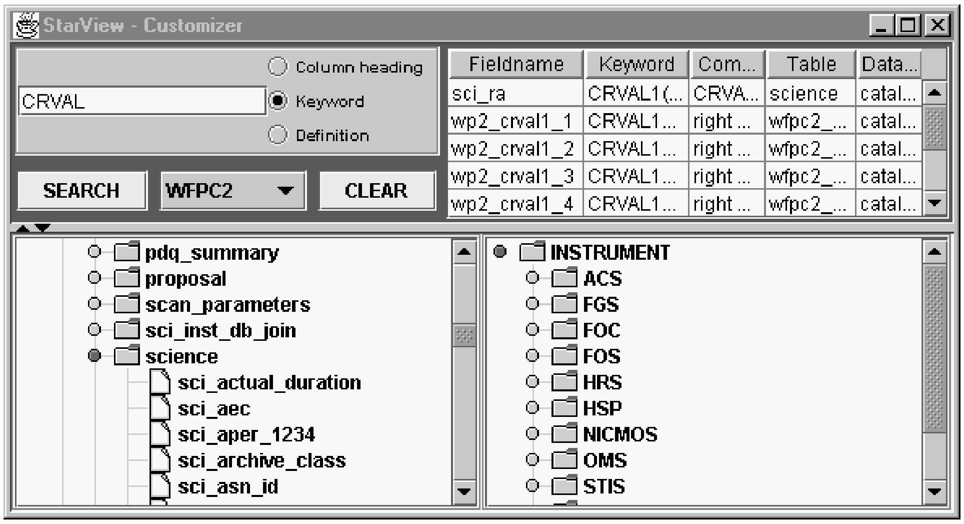

Figure 1. StarView II Customizer.

separate loader class that translates the underlying data into the generic two-dimensional vector. This allows the flexibility to replace or subclass the loader class to handle different data sources such as a flat file or internal memory structure. The rest of the StarView II MVC architecture remains unchanged.

2.2. Selectable, Extensible Data Views

The prototype StarView II view components, list, text field, and table, are extensions of Java Foundation Classes (JFC a.k.a. Swing). The MVC architecture built into JFC provides an ideal backdrop for implementing the StarView II design. Using Java interface classes, view components register interest in data model changes. Once registered with the data model, JFC event processing keeps view components up-to-date with data changes.

View components register as listeners with the base data model through a minimally specified interface class. The data model has no additional knowledge of the characteristics of the listener. Another interface expands this strategy by allowing third-party components to assume the role of a StarView II MVC view. The only imposition, i.e., shared code, on third-party code is implementation of the StarView II MVC component interface. The end goal for StarView II is to allow run-time addition of these components.

2.3. Control

Control, or data modification, is effected by a Query Processor object in response to Search and Navigation user interface (UI) components. The Query Processor interacts with a Database object. Each query form can search using a hard coded query, or, as in the case of HST searches, generate the query at run-time. Automated query generation is a complicated system and is isolated to run on a server allowing the option of turning off or bypassing this mechanism on a per form basis.

Data navigation refers to the ability to step and scan through the database using UI controls. In considering navigation, it is important to discuss the StarView II ability to keep various views in synch via registration with a common data model. For example, as a user steps through records in a list box, the data model sends events to all view component listeners. The view components update their displays reflecting data associated with the record number received in the event.

3. Adaptive Functional Framework

Historically, StarView has been used to browse the HST Data Archive. In setting goals for StarView II, we recognized the need for a more flexible, robust browsing tool and the fact that each user has a different set of functional requirements. We accommodate unique functional requirements by providing a plug-and-play framework in which a user can create a version of StarView tailored to specific needs.

3.1. StarView II Interfaces

The concept of Java interface classes provides the basis for communication between StarView II and third-party components.

- Data pull interfaces allow a component to query StarView II for available attributes and data contained within a form. Using this interface, components can be instantiated at run-time and retrieve any portion of the information contained on a screen.
- Data push interfaces allow a component to push generated data back to a StarView II qualification form.
- MVC interfaces allow a component to register for data model changes. The component receives event notification of a change and then uses the data pull interface to obtain a specific attribute location from the data stream.
- The Observer/Observable interface makes use of a user-defined object class for data transfer between applications. StarView II is required to have prior knowledge of the object's structure. This accommodates more complex command interactions (Dolensky, Mayhew, & Kennedy 1999).

4. Proof of Concept: Third-party component interaction

StarView II's adaptive framework was well illustrated while testing the ease of installation of JIPA (Dolensky et al. 1999) and SkyMap.

SkyMap is a component developed independently at STScI that plots HST observations on an AITOFF projection. The association established between SkyMap and StarView II demonstrates a highly interactive relationship achieved through Java interfaces. Our tests exercise bi-directional interfaces that send data to an external MVC view and receive data from a third-party functional component.

As an MVC component, SkyMap registers as a listener to the main data model and receives data change notifications. SkyMap responds to the event

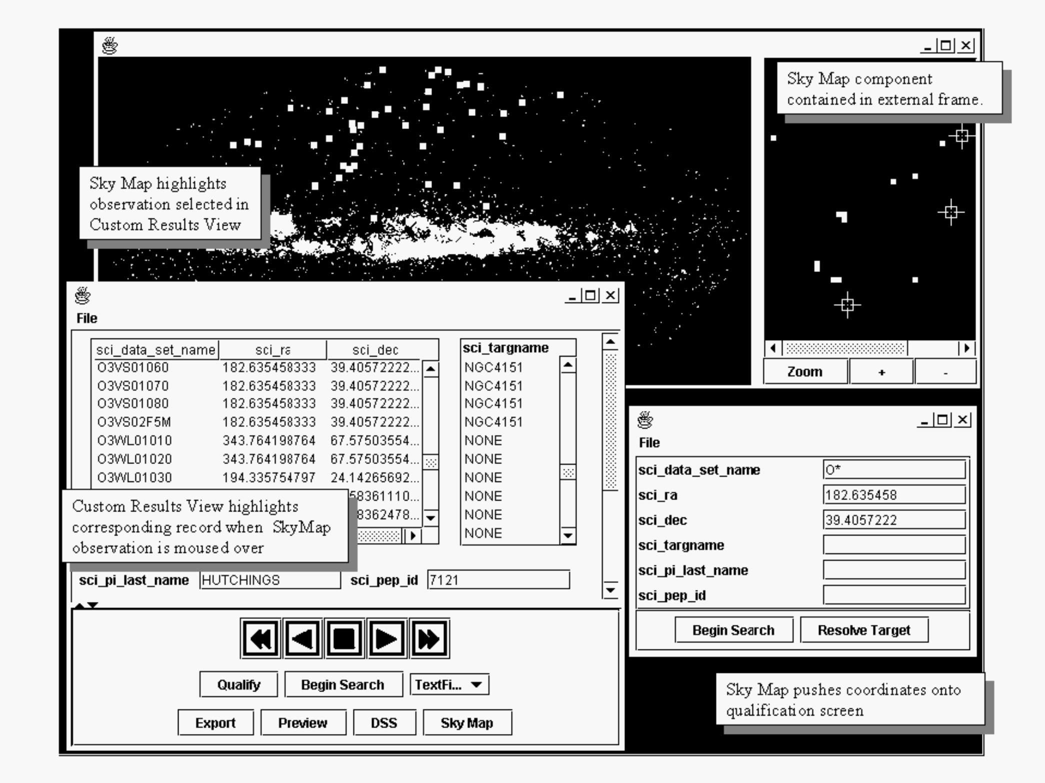

Figure 2. StarView II/SkyMap Interaction.

from the data model by highlighting the location for the current record on the AITOFF projection.

As a member of a group of MVC components associated with a particular data model, SkyMap interacts with other view components. When the mouse enters an observation on the SkyMap, it sends a 'set current record' to the main data model. Other view components receive an event from the model and update accordingly, achieving a seamless interactive graphical display.

Finally, the SkyMap and JIPA tests both exercise the ability to push values into StarView II from an external source. As the user points and clicks on the SkyMap display, or the JIPA preview image, the associated coordinates are pushed into Ra and Dec fields on a qualification screen. These pushed coordinate values can then be used for subsequent searches.

References

Dolensky, M., Mayhew, B., & Kennedy, B. 1999, this volume, 454