ADS' Dexter Data Extraction Applet

Masha Demitriou, Alberto Anzumaci, Günter Eichhorn, Carolyn S. Grant, Michael J. Kurtz, Stephen S. Murray
Harvard-Smithsonian Center for Astrophysics, 60 Garden Street, Cambridge, MA 02138

Introduction

The Astrophysics Data System (ADS, http://adswww.harvard.edu) now holds 1.3 million scanned pages, containing numerous plots and figures for which the original data sets are lost or inaccessible.

The availability of scans of the figures can significantly ease the regeneration of the data sets. For this purpose, the ADS has developed Dexter, a Java applet that supports the user in this process.

Dexter’s basic functionality is to let the user manually digitize a plot by marking points and defining the coordinate transformation from the logical (screen) to the physical (graph) coordinate system. Advanced features include automatic identification of axes, tracing lines and finding points matching a marking points and defining the coordinate transformation from the logical in this process.

This poster demonstrates the operation of Dexter in a little example and discusses some of the architectural issues.

Architecture

The diagram above shows a raw sketch of Dexter’s architecture in a UML-inspired graph. Rectangles symbolize classes, rounded rectangles stand for interfaces, with dashed arrows from their implementation.

In the central position there is the DExtractor class that cares about most of the user interface and contains the logic responsible for the transformation from screen to graph coordinates through the DExtractor class (that controls the text fields for entering the start and end values for the graphical gauge). The DExtractor class is derived from AWT's Frame class.

The panel with the image of the figure is in a different class, ImageWithPoints that also embeds the code changes in both DExtractor (that controls the graphical Gauges and the DExtractor class) and contains some utility functions, for example to acquire the image from the local file. These are stored as bytes for this purpose; the dynamic range from 0 to 127 is more than adequate for the images Dexter deals with, and is also rather useful for file-saving purposes, e.g., in the flood filter used in the PointFinder Recogniser.

Dexter needs to communicate with ImageWithPoints to access the image and to set points to the relevant parts of the figure by using the RealiFinder’s start method. Recognisers will need some sort of user input, e.g., to create a start point for line tracing or a template for point matching. This is done under the control of the Recogniser thread, so that almost all that has to be done from Dexter’s main thread is to call the Recogniser’s start method.

Recognisers do not register themselves automatically with DExtractor, so that some source code changes in both DExtractor (that controls the means bar in which the Recognisers are registered) and ImageWithPoints (that actually starts Recognisers) are necessary when a new Recogniser is written. Given the current scope of the project (about 5000 lines of source code), a more elaborate plug-in scheme did not seem necessary. If Recognisers have adjustable parameters, they can use the Recognition Settings containing both a HasDialog to store the property values and the logic to display a dialog to change them.

All these Recognisers currently implemented (AutoAxisFinder to locate the axes, PointFinder for point matching, LineTracer for automatically digitizing lines), use rather naive algorithms. The computational effort for performing Fourier or Hough transforms on entire images make these approaches currently unattractive in a tool supposed to be interactive like Dexter, in particular given the poor quality of the Java virtual machines on some architectures.

Dexter’s source code is available under the General Public License at http://Dexter-sourceforge.net.

Following the link to Dexter on an article page, one can select the portion of the graph that Dexter is to look at. After the selection, the main Dexter window pops up:

Now the start and end values of the lines are entered. After this, Dexter has all information it needs to transform points from the screen coordinate system to the graph coordinate system and will display graph coordinates in the status line (cf. the last panel in this box).

Just for convenience, the lower point of the vertical gauge is manually adjusted to 0.0.

One could experiment with raising the recognition threshold in the Recognition Options at this point to have Dexter identify the remaining points (or, which would be more effective in this case, increase the figure’s resolution), but since only a few points are missing, we set them manually – the magnifying glass helps.

After pressing the “Show Data” button the data is filled in the text field at the bottom of the window. The “Send Data” and “Save Data” buttons can be used to retrieve the extracted data through the browser.

Operation

Rather than paraphrase the help page for Dexter, we demonstrate Dexter’s operation with an annotated example session.

The next step is to start the automatic axis recognition; in this case, it works quite well.

The diagram above shows a raw sketch of Dexter’s architecture in a UML-inspired graph. Rectangles symbolize classes, rounded rectangles stand for interfaces, with dashed arrows from their implementation. The lines mean that two classes are talking to each other, with the diamond indicating which class is embedded by which.