ADS' Dexter Data Extraction Applet

Markus Demleitner, Alberto Accomazzi, Günther Eichhorn, Carolyn S. Grant, Michael J. Kurtz, Stephen S. Murray
Harvard-Smithsonian Center for Astrophysics, 60 Garden Street, Cambridge, MA 02138

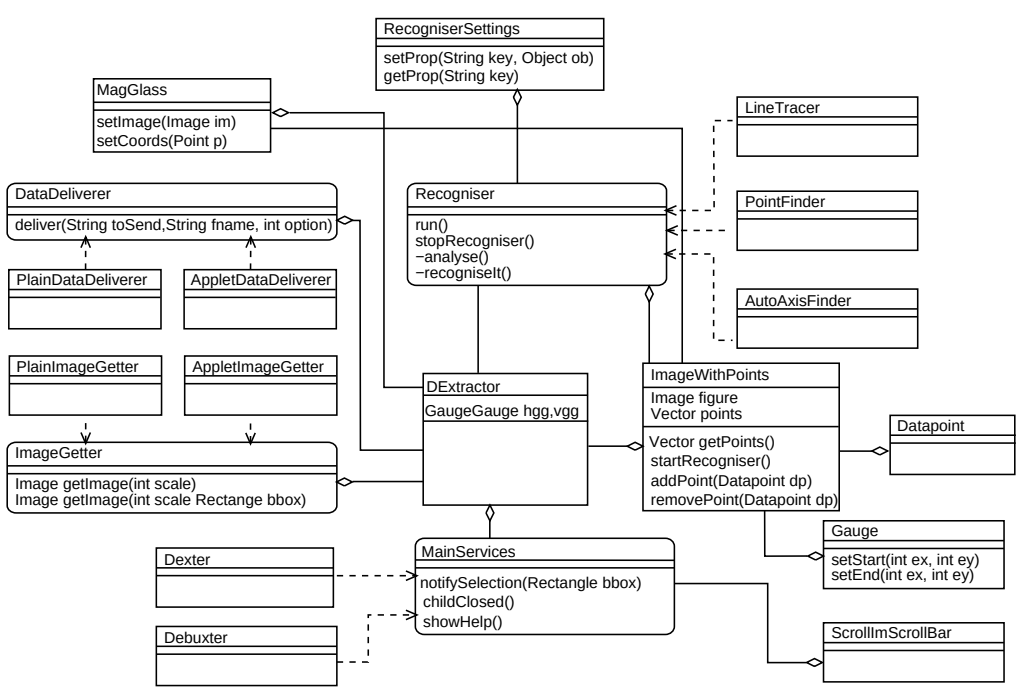
Introduction

The Astrophyiscs 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 template.

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 symbolise 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.

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 GaugeGauge class (that controls the text fields for entering the start and end values for the graphical gauges). 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 classes handling the graphical Gauges and the Datapoints. These latter classes handle clicks on them themselves. For example, the handling of error bars is completely encapsulated within the Datapoint class.

The MainServices interface hides the startup from DExtractor. Two implementations of MainServices exist, Dexter, the applet interface, and the standalone Debuxter that, as the name suggests, was mostly used for debugging. It should not be difficult, however, to implement MainServices that would use, for example, ghostscript, to provide Dexter's capabilities for PostScript articles.

To build such a PostScript-Dexter, one would also need to adjust the implementations of the ImageGetter and DataDeliverer interface, the first accepting requests for scaled and cropped versions of the image, the second delivering the extracted data after the user has requested to save the data or get it sent. Among the implementations of these interfaces, the AppletDataDeliverer is somewhat involved since an (unsigned) applet cannot access the client's disk or portably send data directly to the browser. To save data or display it in a browser window, it is relayed back to the host and tunnelled to the client through a pipe on the host.

The Recogniser interface, derived from Java's Thread class, defines how classes that try to do automatic feature extraction interact with both DExtractor and ImageWithPoints. It also contains some utility functions, for example to acquire the pixels from the graph image. 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 the sign is rather useful for flagging purposes, e.g., in the flood filler used in the PointFinder Recogniser.

Recognisers need to communicate with ImageWithPoints to access the image and to set points or gauges they may have found, but also with DExtractor, giving prompts in the status line and telling it when they are finished. DExtractor needs to know this to re-enable some critical operations that are disabled while a Recogniser is running (changing the resolution of the graph image, sending data). Most Recognisers will need some sort of user input, e.g., to get 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 a call to the Recogniser's start Method.

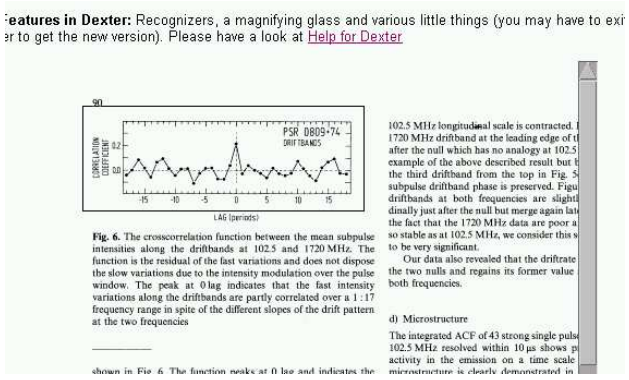
Recognisers do not register themselves automatically with DExtractor, so that some source code changes in both DExtractor (that controls the menu bar in which the Recognisers are registered) and ImageWithPoints (that actually starts Recognisers) are necessary when a new Recogniser is written. Given the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

All three Recognisers currently implemented (AutoAxisFinder to locate the axes, PointFinder for point matching, LineTracer for automatically digitising 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.

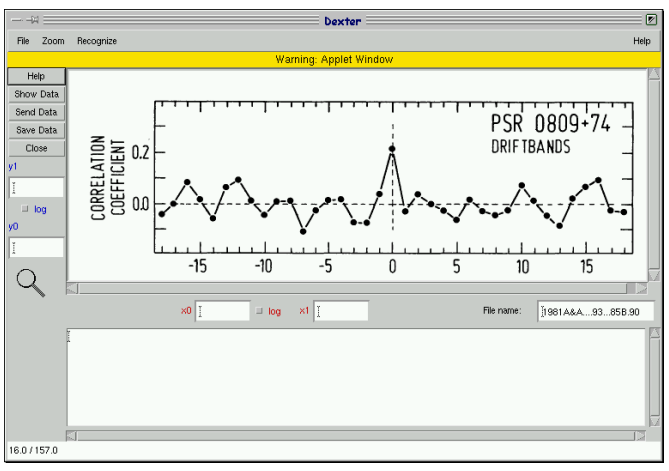
Dexter's source code is available under the General Public License at http://Dexter.sourceforge.net.

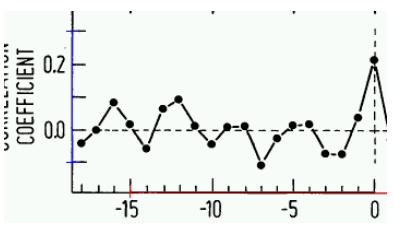
Operation

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



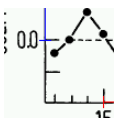
Following the link to Dexter on an article page, one can select the portion of the graph that Dexter is to look at. After the selection, the main Dexter window pops up:

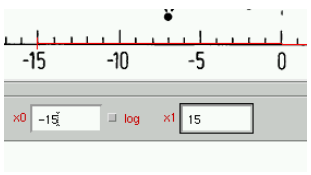




The next step is to start the automatic axis recognition; in this case, it works quite well.

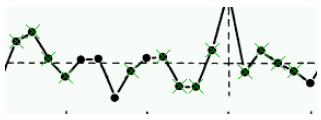
Just for convenience, the lower point of the vertical gauge is manually adjusted to 0.0.

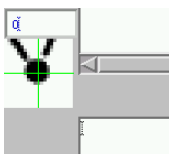




Now the start and end values of the gauges are entered. After this, Dexter has all information it needs to transform points from the screen coordinate system to the graph coordinate system and will display graph coordinates in the status line (cf. the last panel in this box)

These points were found by Dexter after being given a reasonably well-defined point as a template for the automatic point finding.





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 here.

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.