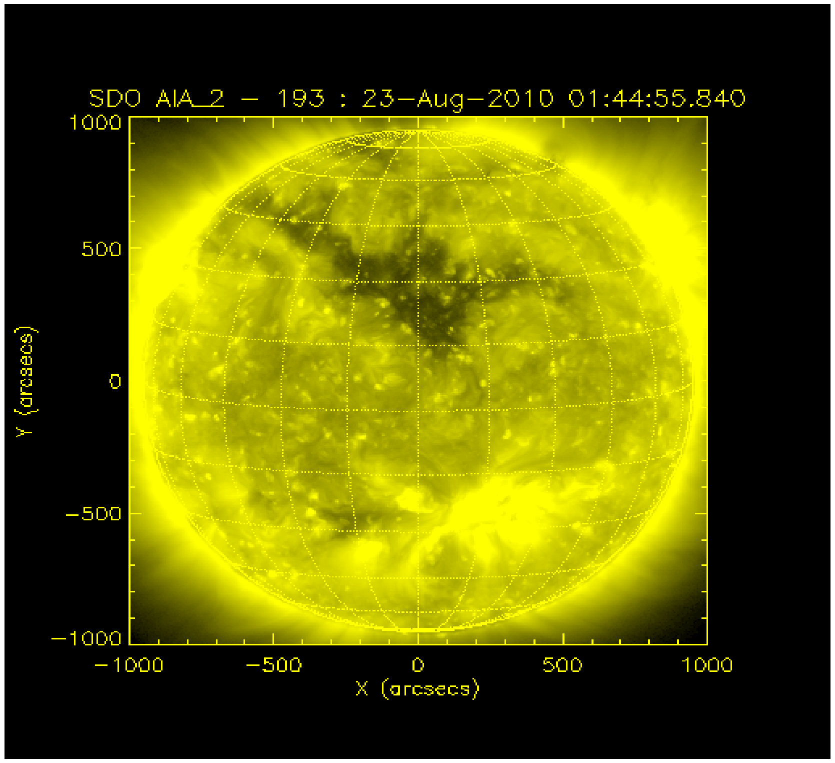
Image Processing Pipelines with Java, Databases and IDL

Leonard Sitongia, Rob Markel, Scott McIntosh

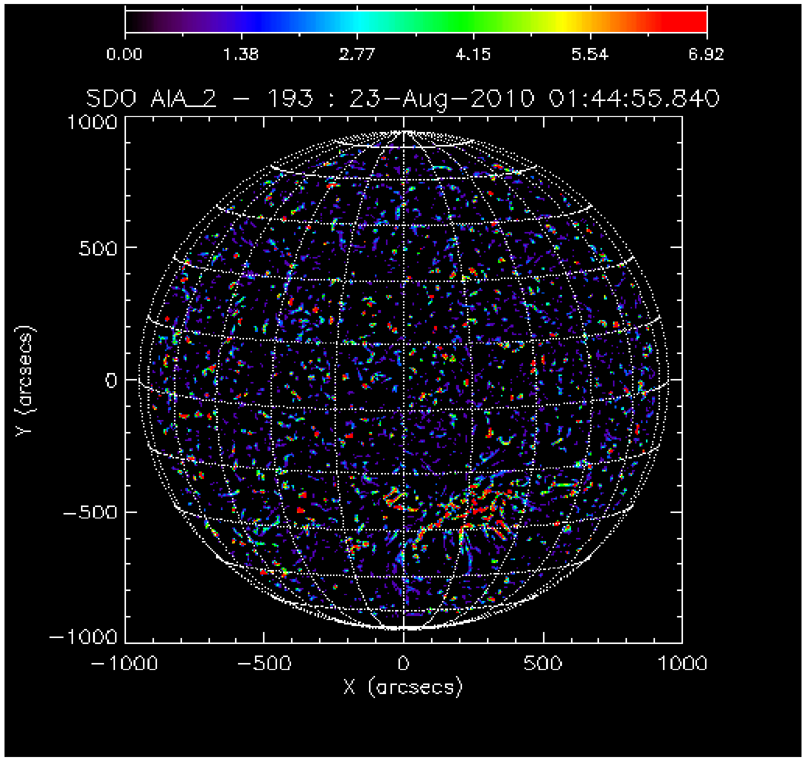
High Altitude Observatory, National Center for Atmospheric Research
Boulder, CO USA

IDL provides specialized image processing procedures through SolarSoft. Java provides a good programming framework and database access. The combination of the two makes sense. ITT provides a way to call IDL from Java and Java from IDL, through their "Bridge". At the High Altitude Observatory, we have developed code to make use of this in a general fashion. Java code drives the pipeline, for file management, date calculations, catalog in metadata. IDL is used for small, instrument-specific calculations. Instrument scientists can develop small code modules to drop into a modern framework developed using modern software engineering methods.

Solar EUV Bright Points



Solar Image from SDO AIA



Sigma Map with BPs

Bright points are one of the solar corona's most ubiquitous features. The coronal Bright Point (BP), observed at X-Ray or Extreme-ultraviolet (EUV) wavelengths, has been associated with small dipolar magnetic fields in the photosphere, sites of local magnetic reconnection in the corona and have been used extensively as a marker of solar differential rotation in the corona . The analysis of a large set of bright point observations can yield important information about the behavior of the coronal plasma and its connection to the photosphere on many spatial and temporal scales, investigating the mechanism that transports magnetic and mechanical energy from the sub-photospheric plasma to support and maintain the Sun's hot corona.

Processing Pipeline

Bright Points are identified through the image processing of calibrated images from many space-based imaging instruments.

The BP Finder performs the image processing to identify BPs in the images. It is a Java program which uses the IDL Bridge to launch an IDL procedure, which performs most of the work. The IDL uses local files or uses the metadata from the Virtual Solar Observatory (VSO) to pull over each full-disk image for each instrument and wavelength and then uses the SolarSoft calibration procedures specific to the instrument. A map of the significance of features is created from the image. Regions of particular significance in sigma are identified as BPs. Characteristics of the Bright Points are measured and image files are saved for the sigma maps. A number of approaches were tried for the storage of resulting metadata in a MySQL database.   Performance problems with MySQL and IDL prevented using the IDL Bridge procedures, to call Java from IDL.   In the approach now in use, XML files are written from IDL and subsequently processed with Java to write to MySQL. The Java XML parser is an external program spawned from IDL.

The Tracker is almost entirely written in Java. Simple blocks of IDL code are called from Java using the IDL Bridge, for instrument-specific time conversion and plotting routines. Passing arguments from Java to IDL is simple to do. Subsequent analysis consists of mining the database in various ways. BP characteristics can be statistically analyzed. Most importantly, BPs are tracked from image to image, thereby linking BPs in the database to each other through tracking, allowing further analysis of BPs and their movement and lifetimes. This data mining Tracker is written in Java for performant database access.

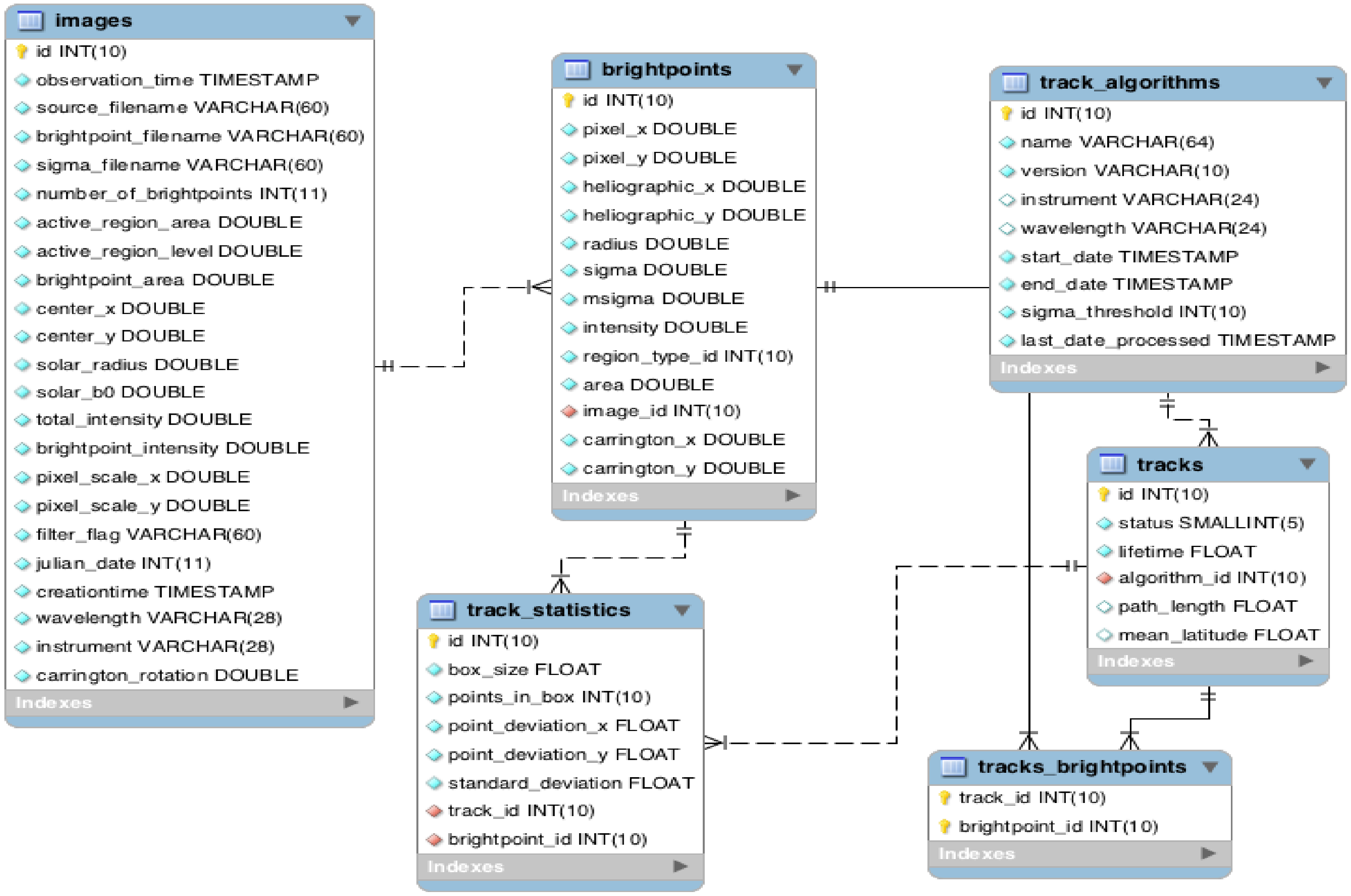
Data Management and Model

SOHO EIT provides EUV images from three wavelengths for over 14 years. Over 36,000 images have been analyzed. STEREO EUVI is a pair of newer instruments, also providing three wavelengths, and has been operating for four years now. Over 400,000 images are available. Each images has 100-200 BPs. Collectively, these instruments result in about 70 million BPs to track. In addition, we plan to analyze images from the Hinode and Yohkoh spacecraft. The newest spacecraft, the Solar Dynamics Observatory (SDO), recently launched, is producing a mammoth amount of data. Our approach for SDO will probably be for studies over relatively short time spans.

Smaller image datasets are stored on local disks. Larger ones are acquired over the network from the Virtual Solar Observatory (VSO) as they are analyzed and are not stored locally. Output from the Finder is stored locally in all cases. Those images are used for making movies and checking operations.

BPs and subsequent Tracks of their motion across the solar disk are characterized by a number of parameters, all of which are stored in the database. The BPs are collected once, but data mining for the Tracks can be performed repeatedly with different sets of conditions. These conditions are parameterized under the name of an algorithm and so tag different sets of Tracks from the same set of BPs.

Relationships in the database are properly maintained for integrity and so that subsequent analysis of the Tracks allows posing new questions about characteristics of the BPs.


images
id INT(10)
observation_time TIMESTAMP
source_filename VARCHAR(60)
brightpoint_filename VARCHAR(60)
sigma_filename VARCHAR(60)
number_of_brightpoints INT(11)
active_region_area DOUBLE
active_region_level DOUBLE
brightpoint_area DOUBLE
center_x DOUBLE
center_y DOUBLE
solar_radius DOUBLE
solar_b0 DOUBLE
total_intensity DOUBLE
brightpoint_intensity DOUBLE
pixel_scale_x DOUBLE
pixel_scale_y DOUBLE
filter_flag VARCHAR(60)
julian_date INT(11)
creationtime TIMESTAMP
wavelength VARCHAR(28)
instrument VARCHAR(28)
carrington_rotation DOUBLE
Indexes
brightpoints
id INT(10)
pixel_x DOUBLE
pixel_y DOUBLE
heliographic_x DOUBLE
heliographic_y DOUBLE
radius DOUBLE
sigma DOUBLE
msigma DOUBLE
intensity DOUBLE
region_type_id INT(10)
area DOUBLE
image_id INT(10)
carrington_x DOUBLE
carrington_y DOUBLE
Indexes
track_algorithms
id INT(10)
name VARCHAR(64)
version VARCHAR(10)
instrument VARCHAR(24)
wavelength VARCHAR(24)
start_date TIMESTAMP
end_date TIMESTAMP
sigma_threshold INT(10)
last_date_processed TIMESTAMP
Indexes
tracks
id INT(10)
status SMALLINT(5)
lifetime FLOAT
algorithm_id INT(10)
path_length FLOAT
mean_latitude FLOAT
Indexes
track_statistics
id INT(10)
box_size FLOAT
points_in_box INT(10)
point_deviation_x FLOAT
point_deviation_y FLOAT
standard_deviation FLOAT
track_id INT(10)
brightpoint_id INT(10)
Indexes
tracks_brightpoints
track_id INT(10)
brightpoint_id INT(10)
Indexes

Bridging between Java and IDL

The Interactive Data Language (IDL, *http://www.ittvis.com/ProductServices/IDL.aspx*) is an interpreted programming language for image processing and data computation. It is great for small tasks, developing analysis algorithms interactively and displaying small image sets. Much of the software used in solar physics uses IDL through the SolarSoft (*http://www.lmsal.com/solarsoft/*) collection of IDL procedures. IDL is not practical for large, non-interactive database-driven processing efforts. Attempts to accomplish such efforts with IDL tend to result in unwieldy procedural programs which become difficult to understand and maintain. Performance has been unacceptably low for the bulk of the analysis of the BP database.

Java (*http://www.oracle.com/technetwork/java/*) is a great language for engineering modern software. Performance has been reasonable. The results have been object oriented code which is easy to comprehend, maintain and modify.

Bridging from IDL to Java has proven to be problematic. On the other hand, Bridging from Java to IDL has proven to be acceptable and provides a way to use chunks of IDL code. The Bridge is simply launched in Java with

```
connection = new java_IDL_connect();
connection.createObject();
IDLConnectionReader outputListener = new IDLConnectionReader();
connection.addIDLOutputListener((JIDLOutputListener) outputListener);
```

This creates a IDL process daemon which listens for command requests, such as

```
idl.execute(command);
```

The Tracker's use of the Bridge is exemplified by

```
idl.execute("deltaLongitude = "
        + "double(diff_rot("+differenceTimeInDays+","+latitude+","+keyword+"))");
double deltaLongitude = idl.readDouble("deltaLongitude");
```

where the IDL procedure to calculate solar differential rotation is called and the value is returned to Java.

A further example is in calling an IDL plot procedure from Java

```
public void doBrightpointsPlot(float[][] numbers, Date startDate, String title) {
        idl.setString("tstart", parseTime(startDate));
        idl.setString("title", title);
        idl.setFloat2DArray("numbers", numbers);
idl.execute("do_brightpoints_plot, numbers, tstart, title");
}
```

In this project, it has proven to be possible to use IDL while working in a modern, object-oriented software engineering environment for scientific image processing.

For questions, contact sitongia@ucar.edu