
The VOTech Project

# Design Study 6: Data Exploration Preliminary Study Report

## January 2007

# Table of Contents

Table of Contents....2
Table of Figures....4
1. Introduction....6
1.1 Executive Summary....6
1.2 VOTECH Project Background....7
1.2.1 Project Summary....7
1.2.2 Project Objectives....8
1.2.3 Objectives of the VOTECH Project:....9
1.2.4 Scope of DS6....9
1.3 Outline of the remainder of this report....10
2. Data Mining and Astronomy....10
2.1 Introduction to data mining....10
2.1.1 Automation....11
2.1.2 Data Mining Tasks....12
2.2 Data mining in astronomy....12
*2.2.1 Efficient Photometric Selection of Quasars from the SDSS*....12
*2.2.2 Photometric redshifts in the nearby universe*....13
2.3 Summary and Conclusions....13
3. Visualization and Astronomy....14
3.1 Introduction....14
3.2 Image visualization with Aladin....14
3.2.1 Basic functionalities....14
3.2.2 Advanced functionalities....15
3.3 Visualisation of Tabular Data in TOPCAT....16
3.3.1 Introduction....16
3.3.2 Design....16
3.3.3Visualisation Features....18
4. DS6 Science Drivers....23
5. DS6: The VOTECH Data Exploration Design Study....24
5.1 Introduction....24
5.2 Survey of existing software....24
5.3 Interoperability of client-side tools....24
5.3.1 Introduction....24
5.3.2 Background....25
5.3.3 Previous Work....25
5.3.4 Technical Overview....26
5.3.5 PLASTIC in action....26
5.3.6 Technical Details....30
5.3.7 PLASTIC Messages....31
5.3.8 Example Messages....31
5.3.9 Current Status and Future Work....32
5.4 VO-enabling existing data exploration software....32
5.4.1 TOPCAT....32
5.4.2 AstroWeka....35
5.4.3 VisIVO....40
5.4.4 Aladin: image display....44
5.4.5 Aladin: metadata visualization....52

5.5 Addressing scalability issues ..... 57
5.5.1 Scalable access to tabular data ..... 57
5.5.2 Cached sufficient statistics ..... 69
5.5.3 Reducing the amount of data to be visualized: Eirik ..... 71
5.5.4 The VOTechBroker: parameter sweeps on the Grid ..... 77
6. Conclusions and Future Work ..... 82
References ..... 83

# Table of Figures

Figure 1: The conceptual structure of VOTECH....8
Figure 2: Aladin in multiview mode....15
Figure 3: TOPCAT Scatter Plot window....20
Figure 4: TOPCAT 3D Scatter Plot window....20
Figure 5: TOPCAT Spherical Plot window....21
Figure 6: TOPCAT Line Plot window....21
Figure 7: TOPCAT Histogram window....22
Figure 8: TOPCAT Density Plot window....22
Figure 9: Sending a VOTable to TOPCAT with PLASTIC....26
Figure 10: Reduced Proper Motion Diagram....27
Figure 11: Updating menus once VisIVO has registered with the Hub....28
Figure 12: Visualizing the 4D plot in VisIVO....29
Figure 13: Overplotting in VisIVO the white dwarf candidates selected in TOPCAT....30
Figure 14: The Explorer window in AstroWeka....36
Figure 15: Accessing MySpace from AstroWeka....37
Figure 16: Finding and using conesearch services with AstroWeka....38
Figure 17: Using AstroWeka and TOPCAT together, via PLASTIC....39
Figure 18: VisIVO visualizing an HDF5 dataset....41
Figure 19: An example of an isosurface view in VisIVO....42
Figure 20: An example of interoperability between VisIVO, Aladin and TOPCAT....43
Figure 21:The PLASTIC preferences window in Aladin....46
Figure 22: The radio antenna icon in Aladin reflects PLASTIC status....47
Figure 23: Possible states of the PLASTIC icon in Aladin....47
Figure 24: Broadcasting an Aladin catalogue plane with PLASTIC....48
Figure 25: Sending a catalogue URL from Aladin to a PLASTIC application....48
Figure 26: Exchanging data between Aladin, TOPCAT and VisIVO....49
Figure 27: Sending URLs of spectra from Aladin to VOSpec and SPLAT....50
Figure 28: The eSTAR PLASTIC-based event monitor....51
Figure 29: Linked views between TOPCAT, Aladin and VisIVO....51
Figure 30:An example of an Aladin plugin using PLASTIC....52
Figure 31: List of datasets and metadata for an image displayed in Aladin....53
Figure 32: Fields of Views of observations plotted on top of a preview imagein Aladin....53
Figure 33: The datatree for a cube, with the cube preview displayed....55
Figure 34: The data retrieval page....55
Figure 35: Lambda plane access....55
Figure 36: Selection of spectrum position....55
Figure 37: Spectrum visualization....55
Figure 38: Characterisation of a dataset: output from the Aladin image server in Datascope....56
Figure 39: A k-d tree for four elements....69
Figure 40: Using a k-d tree to accelerate nearest neighbour search....70
Figure 41: Some techniques for column reduction....71
Figure 42: Some techniques for row reduction....72
Figure 43: The client-server data flow in the Eirik design....74
Figure 44: The UCD tree view in Eirik....75
Figure 45: Histograms in Eirik....76
Figure 46: A Mutual Information/Covariance plot in Eirik....77

Figure 47: VOTB: Logical architecture.....79
Figure 48: VOTB implementation.....80
Figure 49: JavaSpace and the distribution of submission workers over a LAN or SAN .....81

# 1. Introduction

This is a preliminary report by the Design Study 6 (DS6) team of the VOTECH project. It covers the work undertaken by the DS6 team from 1 January 2005 to 31 December 2006 and outlines the plans for the remainder of the project, as well as the scope of the *First Data Exploration Prototype Release*. This prototype release, due in March 2007, is one of the formal deliverables from VOTECH, along with the *Data Exploration Study Report*. It was decided by the VOTECH Consortium Board that it would be premature to release the *Study Report* on its original due date in December 2006, given the current status of the DS6 team's work, and that it would be preferable to release a *Preliminary Design Study Report*, covering the period to the end of 2006, which will be followed later by a final version of the *Study Report*. This has the advantage of apprising the community of the significant progress made by the DS6 team to date, and enabling their input into the detailed planning of the remainder of the DS6 workplan, while ensuring that the deliverables from DS6 record all its achievements, and not just those before the end of 2006.

This report was compiled by the following present and former members of the VOTECH team: Ugo Becciani, Thomas Boch, François Bonnarel, Marco Comparato, Richard Holbrey, Bjorn Larsson, Giuseppe Longo, Bob Mann (DS6 Lead and report editor), Bob Nichol, Clive Page, Garry Smith, Antonino Staiano, John Taylor, Mark Taylor and Brian Walshe. Its structure is intended to foreshadow that of the final *Study Report*, with the result that, since the full DS6 workplan has not yet been completed, it is more complete in some sections than in others, as indicated in the text.

## 1.1 Executive Summary

The aim of DS6 is to complete all technical preparatory work necessary to enable effective data exploration within the future European Virtual Observatory (EuroVO[1]). This will involve **assessment of a range of data mining and visualization algorithms and packages**, with a view to determining how they can be run as **distributed services**, how they can be made **VO-compliant** and how they can be extended to **extremely large datasets**. On the assumption that these studies are successful, the DS6 team will proceed to actual component designs, trial implementations and standards development.

In the period to the end of 2006, the DS6 team has made significant progress towards all of these objectives:

- **Assessment of existing software**: an initial Software Survey was produced, as part of the design process behind the Eirik prototype tool (see Section 5.5.3). This will be completed for inclusion in the final version of the Study Report.
- **Running distributed data mining services**: the VOTECHBroker (see Section 5.5.4) has been developed as a way of facilitating the deployment of data mining jobs in a distributed, Grid- or cluster-based environment.
- **VO-compliance**: the DS6 team developed a document describing what VO-compliance would mean for a data exploration tool and detailing how to achieve it, and this has been successfully applied in making a number of existing tools – e.g. TOPCAT (Section 5.4.1), Weka (5.4.2) and VisIVO (5.4.3) – compliant with VO standards.
- **Scalability**: as discussed in Section 5.5, several approaches to meeting the scalability challenges facing VO data exploration have been followed to date, notably improving

access to large tabular datasets through column-ordered storage and memory mapping (Section 5.5.1) and selecting useful rows or columns in dataset through the use of the Eirik tool (Section 5.5.3) to reduce the amount of data to be visualized.

Perhaps the greatest success of the DS6 work to date has been the development of the PLASTIC protocol (see Section 5.3) for making client-side tools interoperable. This has been adopted with enthusiasm by the developers of many of the most prominent existing tools, acclaimed by research astronomers as providing something they can immediately use, and is now being translated into an IVOA[2] standard, under the aegis of the new Applications Working Group[3].

For the remainder of the life of the VOTECH project, the DS6 team will focus on the delivery of robust prototype software to enable users to gain experience of data exploration within the developing global VO and on the further development of IVOA standards which encapsulate the best practice and insights emerging from the DS6 R&D programme. In terms of that R&D programme, work to date has centred on making existing client-side tools VO-enabled and interoperable, while the remaining effort will focus on scalable data exploration, using sophisticated data structures and algorithms, or taking advantage of the computational resources on offer in the Grid.

# 1.2 VOTECH Project Background

## 1.2.1 Project Summary

VOTECH is a Design Study aimed at completing all technical preparatory work necessary for the construction of the European Virtual Observatory (EuroVO). The concept of the Virtual Observatory (VO) is that all the world's data should feel like they sit on the astronomer's desk top, analysable with a user-selected workbench of tools, made available through a standard interface. Internationally this is set to transform and re-structure the way astronomy is done. EuroVO is a specifically European implementation of this idea, and will produce a world-leading infrastructure providing a unified virtual data resource and the ability to perform complex data discovery and manipulation tasks across the whole range of astronomy. Access to data and tools will be equally good across Europe, regardless of location. This will require establishing an alliance of data centres, and a VO facility centre in support of the community, but crucially requires the construction of an infrastructural glue of software components, in the context of rapidly evolving background developments in IT and the Grid.

The VOTECH project aims specifically at feasibility studies and design work aimed at integrating such new technologies into the EuroVO. Key IT advances to build on are in intelligent resource discovery (ontology and the semantic web), data mining, and visualisation capabilities. These will be integrated via global astronomical interoperability standards coupled with the latest distributed Grid computing services. Additionally this project covers design and preparatory work to ensure that data from the major European telescopes and facilities (as represented by the Opticon[4] and RadioNet[5] networks) is fully accessible through the EuroVO, and where required, is able to offload mass scale computational process onto the EGEE[6] backbone.

In summary, VOTECH will lay the technical foundations of the EuroVO, a European infrastructure to revolutionise the scientific process.

## 1.2.2 Project Objectives

The top-level objective of the VOTECH proposal is to complete all technical preparatory work necessary for the construction of the European Virtual Observatory.

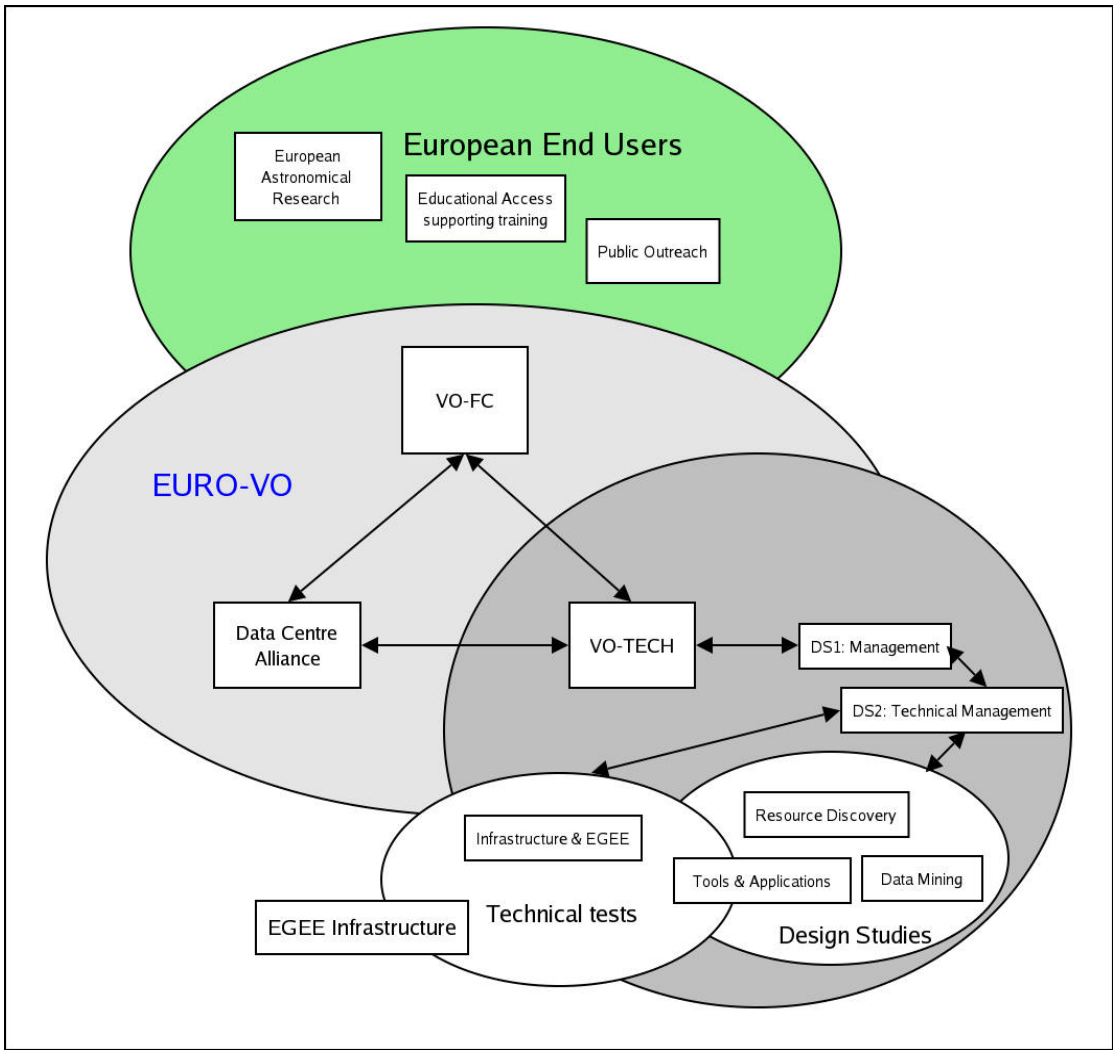


**Figure 1: The conceptual structure of VOTECH**

Figure 1 shows the conceptual structure of VOTECH, and how it relates to EuroVO as a whole, the various classes of user, and the general astronomical infrastructure. The VOTECH preparatory work needs to link closely with the work of the Data Centre Alliance (DCA[7]) and VO Facility Centre (VOFC), with the final construction of EuroVO following the VOTECH design study kept in mind. The work also takes place in the context of extensive developments - new algorithms, technologies, and protocols - in academic and commercial IT, and especially of course in generic Grid middleware. This work will not be repeated. The job needed is to assess these developments and design astronomy-specific modules based on them. The working links of the project partners with this external world are excellent. Several of the VOTECH partners have active working relationships with the academic and commercial IT communities, and the project will work with EGEE as an exemplar application area. It is also worth noting that Astronomy in general, and the VO projects in particular, have attracted attention as leading edge but pragmatic exemplars of the new e-Science approach - for example some of the VOTECH co-Is have been invited to talk at Bio-Informatics meetings, as well as general Grid meetings.

## 1.2.3 Objectives of the VOTECH Project:

1. To assess new technologies and study the feasibility of their incorporation in EuroVO
2. To create designs of new infrastructure components based on those new technologies
3. To create designs of science user tools and data mining services
4. To develop trial versions of new infrastructure components, tools, and data mining services and to test them
5. To decide what new interoperability standards are required, and to define those standards with international partners
6. To liaise with the larger EuroVO structure, gaining refreshed versions of science functionality and architecture, and feeding back component test results, designs, and trial components for demonstration suites.
7. To liaise with computer science, IT industry, and related applications projects in order to mesh with larger standards and to save work wherever possible

## 1.2.4 Scope of DS6

The original VOTECH proposal described the aim of DS6 as being to complete all technical preparatory work necessary to enable effective data exploration within the future European Virtual Observatory (EuroVO). This will involve **assessment of a range of data mining and visualization algorithms and packages**, with a view to determining how they can be run as **distributed services**, how they can be made **VO-compliant** and how they can be **extended to extremely large datasets**. On the assumption that these studies are successful, DS6 will proceed to actual component designs, trial implementations and standards development.

In more detail, elaborating on each of the terms in bold above in turn:

- **Assessment of algorithms and packages**. The DS6 team will undertake a Software Survey to summarise the assessment of relevant, existing software, in preparation for its *Data Exploration Study Report*.

- **Running tools as distributed services**. The DS6 team will prototype the wrapping of existing command line tools using the AstroGrid Common Execution Architecture (or its derived IVOA standard) to expose them as web services which can be run within the developing VO infrastructure. DS6 shall assess whether some data exploration tasks are sufficiently compute-intensive that they need to be run as compute-grid jobs, and will determine the impact that has on the VO.

- **VO-compliance**. Existing tools will be made VO-compliant by making them conform to agreed (e.g. IVOA) standards for registry metadata, workflow and use of distributed storage. Conversely, the work of DS6 will exercise existing standards (e.g. VOTable[8] as a transport mechanism for tabular datasets) and may drive the development of new standards within the IVOA framework.

- **Extension to extremely large datasets**. In addition to assessing the possible requirements for parallel/distributed computations within the VO, the DS6 team will study the scalability issues essential to the effective exploration of data sets of large volume and/or high

dimensionality in the future EuroVO. This may include the use of specialized data structures (e.g. k-d trees[9], etc), the deployment of data exploration services directly on databases at the data centre and the generation of visualizations on the server side.

To attain these goals the DS6 team will follow a twin-track approach - the first focussed on short-term goals that will feed directly into software releases of VOTECH's early years, and the second addressing harder issues which will only yield solutions over the lifetime of the project. The work of DS6 must be science-driven, feeding on requirements from the VOTECH Science Team and also exploiting the scientific motivation and expertise of its own team members.

## 1.3 Outline of the remainder of this report

This report presents a certain amount of background material before moving on to describe the work carried out within DS6 to date. In Section 2, we introduce data mining, and outline the range of tasks usually understood as falling within its scope, as well as the common list of issues which must be faced in performing any data mining analysis. We follow that with two cases studies of astronomical data mining analyses, and Section 2 ends with the conclusions drawn as to what are the significant issues which DS6 should study within the data mining domain. Section 3 describes the kinds of visualization which are already performed by astronomers, as exemplified by the functionality delivered by two tools – Aladin [10] and TOPCAT [11] – developed by staff associated with DS6[1]. In Section 4 we briefly describe the science drivers behind DS6, which centre on the vast federated dataset which will become available through the VO. Section 5 summarises DS6 work to date. After an introduction and an outline of the DS6 survey of existing software, we come to a discussion of the requirements for interoperability between client-side data exploration tools, and our solution to that problem, the PLASTIC [14] protocol. Section 5.4 details a series of case studies in VO-enabling data exploration tools, and that is followed by a summary of DS6 work to date addressing the scalability issues inherent in VO data exploration. Finally, Section 6 presents conclusions from the DS6 work so far, and an outline of the work plan for the remainder of the project: in the final *Study Report*, this section will include a more detailed discussion of the data exploration requirements of a working EuroVO and recommendations as to how to meet them, based on the DS6 design study, but it is premature to include too much along those lines here.

# 2. Data Mining and Astronomy

## 2.1 Introduction to data mining

Data mining is a relatively new discipline which has arisen from the need to study information contained in very large databases. Data mining evolved from Exploratory Data Analysis in

[1] Note that, throughout this report, we do focus on tools whose developers have a strong relationship with VOTECH. This does not mean that comparable functionality is not available elsewhere – for example, GAIA [12] and DS9 [13] offer image display capabilities broadly similar to that of Aladin – nor that we do not seek to include tools developed elsewhere interoperably into the VO, but only that, with finite resources, we have concentrated our efforts to date on those tools over whose development we have most influence. The example of PLASTIC (see Section 5.3 & 5.4) illustrates how readily ideas generated within VOTECH can be adopted by tool developers with no direct relationship to the project.

statistics, and the line between the two is still quite blurred, making use of traditional statistics augmented with machine learning and database technology. It can be defined as:

*The analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner*[2].

It is the nature of the data being analyzed that differentiates data mining from traditional statistics and machine learning. Because of the large volume of data, new problems must be taken into consideration. These range from basic technical issues, such as how the data will be stored and processed, to more fundamental issues, such as how to differentiate between chance patterns in the data and real discoveries.

Data mining operations typically use observational data which have already been collected independently, e.g. astronomical survey data available in the VO. This allows new discoveries to be made using old data, but requires caution on the part of the data miner. Care must be taken to ensure that the available data are representative of the general population, and the large volume can make this difficult to establish.

The data can be noisy, containing errors and missing values. While most survey data have been preprocessed and cleaned, different surveys use different methods and calibrations, etc. The heterogeneous nature of the data must be taken into account, and will often require additional preprocessing and transformation to ensure like is being compared with like.

### 2.1.1 Automation

Data mining uses automated techniques to search through datasets to find patterns and develop models. The basic algorithmic process contains the following components:

- **Model or Pattern Structure**: determining the underlying structure or functional forms we seek from the data.
- **Score Function**: judging the quality of a fitted model.
- **Optimization and search method**: optimizing the score function and searching over different model and pattern structures.
- **Data Management Strategy**: handling data access efficiently during the search/optimization.

Most machine learning algorithms have been developed under the assumption that the data being analyzed can be loaded into random access memory, where they can easily be accessed. Often, with large data sets, this is not possible and the data may need to be stored on disk, or even another computer on the network. Traditional machine learning techniques do not scale well when they have to use this slower storage, and their running time becomes an issue.

[2] This quote is from the standard text[15] by Hand, Mannila and Smyth (2001), from which we have taken much of the material in this Introduction. Other recommended introductions to data mining are Andrew Moore's Statistical Data Mining Tutorials[16] and the book *Data Mining: Practical Machine Learning Tools and Techniques* by Witten and Frank [17] which describes the Weka[18] toolkit.

There are several ways to deal with this problem, generally by adapting the algorithms operate. One method is to use on-line algorithms which only need to read through the data set once. Another is to use cached sufficient statistics, stored in a data structure such as a k-d tree, which can be used to reduce the need to access the data set to perform any calculations. High performance and high throughput computing can further speed up the process. Grid concepts, such as moving the computation to the data can help reduce access times.

### 2.1.2 Data Mining Tasks

Operations typical of data mining can be divided broadly into five categories: *exploratory data analysis*, *descriptive modeling*, *predictive modeling*, *discovering rules and patterns*, and *retrieval by content*.

- **Exploratory Data Analysis**: the initial analysis undertaken when the data are first encountered. Interactive and visual techniques are used to give the researcher a feel for the data.
- **Descriptive Modeling**: including techniques such as density estimation (modeling the overall probability distribution of the data) and clustering (breaking the data into naturally occurring groups).
- **Predictive Modeling**: attempt to estimate the value of one variable based on the known values of the other variables. Predicting continuous variables is called *regression*, prediction nominal variables is called *classification*.
- **Discovering Rules and Patterns**: involves searching for combinations of values that frequently occur together. Often combinations that occur very rarely in a data set can also be of interest.
- **Retrieval by Content**: involves searching the dataset for patterns that are similar to one supplied by the user. A typical application in astronomy would be searching for images that match a sample.

## 2.2 Data mining in astronomy

Astronomers have, of course, long sought rules and patterns in their data and created models enabling them to predict the value of one quantity on the basis of others. However, it is only relatively recently that they have started to take full advantage of the advances in recent decades in the statistical and computer science communities which have given rise to the new discipline of data mining. In what follows we present two recent cases studies of the application of such techniques in astronomy.

### *2.2.1 Efficient Photometric Selection of Quasars from the SDSS*

A long-standing problem in extragalactic astronomy has been the creation of clean and complete samples of quasars. By definition, QSOs appear star-like in the optical/near-IR data in which they have been most commonly found, and it has proven difficult to use simple cuts in colour-colour plots to generate quasar samples free from significant contamination by stars. Extensive spectroscopic follow-up is needed to distinguish stars from quasars, and this necessarily greatly lengthens the time needed in preparing the quasar sample for analysis.

This problem faced those members of the Sloan Digital Sky Survey (SDSS[19]) consortium wanting to create a large quasar sample from the SDSS photometric survey. Richards et al (2002)[20] describe a fairly complex procedure they derived on the basis of simulated properties of star and quasar populations for targeting quasar candidates for spectroscopic confirmation, but conclude that its efficiency would only be about 65%: i.e. for every three spectra of quasar candidates taken, only two would yield quasars, and the third candidate would be revealed to be a star. Given the size of the quasar candidate lists that can be generated from large sky surveys like the SDSS – Richards et al were producing targets for a spectroscopic campaign to gain 100,000 quasar spectra – this inefficiency can cause a very substantial waste of telescope time.

In a subsequent paper (Richards et al 2004[21]), many of the same authors – supplemented by data mining experts from the interdisciplinary Pittsburgh Computational AstroStatistics Group – approach the same problem in a different way. They use the distribution of known stars and known quasars (including many from the spectroscopic follow-up of the Richards et al, 2002, candidates) in the four-dimensional space defined by the SDSS (u-g), (g-r), (r-i) and (i-z) colours to create a non-parametric Bayesian classifier which computes the probability that a given source is a quasar, given its location in the 4D colour space. This is conceptually quite straightforward, but computationally challenging on the face of it, since it requires the determination of the density of stars and quasars in the 4D space, and density estimation is, naively, an $N^2$ operation, making it prohibitively expensive for a dataset as large as the SDSS DR1. So, Richards et al make use of a space-partitioning tree representation of the dataset, which allowed them to compute the necessary density very quickly.

The result of this procedure was a catalogue of $10^5$ quasars, created from photometric data alone, which was estimated to be >95% reliable – i.e. less than one in twenty of the "quasars" is actually a star. This is comparable to what can be achieved via spectroscopic confirmation, so, the adoption of this classifier from the data mining world effectively removed the need for spectroscopic follow-up of quasar candidates completely.

### *2.2.2 Photometric redshifts in the nearby universe*

Our second example also involves SDSS data, this time analysed using Astroneural [22], a toolkit implementing neural networks and similar techniques, which will be VO-enabled as part of the DS6 work programme. The goal of D'Abrusco et al (2007)[23] is to determine photometric redshifts for SDSS galaxies, which they do by training Multi-Level Perceptrons (MLPs) for two redshift ranges ($z<0.25$ and $0.25<z<0.5$) using the SDSS spectroscopic sample of ~550,000 galaxies. The advantage of this neural network method over other more "astrophysical" methods, such as template fitting, is that it leads to result sets with fewer catastrophically misestimated redshifts and less evidence for systematic errors.

## 2.3 Summary and Conclusions

The previous section described two situations where techniques from the data mining/machine learning literature were successfully applied to astronomical problems involving large datasets. In both cases, the teams undertaking the analyses contained researchers with considerable expertise in those techniques – either computer scientists or astronomers who have specialised in the application of such techniques to astronomy – so one challenge for DS6 is to see whether it is

possible to deliver the kinds of functionality used in these two projects in a such a fashion that it can be used (and not misused) by those less experienced. Another factor common to both is the use of large volumes of sky survey data, suggesting that scalability is another concern for DS6.

# 3. Visualization and Astronomy

## 3.1 Introduction

The visualization of images and multivariate data play an important role in astronomy. In this Section we outline the range of functionality offered by two of the leading visualization tools - Aladin and TOPCAT. In later sections we shall discuss how these tools are VO-enabled and made interoperable using PLASTIC, so that their capabilities can be used in tandem.

## 3.2 Image visualization with Aladin

### 3.2.1 Basic functionalities

#### 3.2.1.1 Pixel mapping

Aladin displays images as grey-level bitmaps with 8bits/pixel, or RGB images using 3 bytes/pixel. There is a default mapping of the original data to these bitmaps and it is possible to change the mappings. The mapping is defined via two fundamental quantities: cuts and mapping laws. The default law is linear, while default cuts are computed on the fly by an "autocut mechanism". It is possible to change the law, the cuts or both. Possible laws are: linear, log, square root, square. Cuts can be changed to any value, including the extrema of the original data. It is possible to reverse the mapping and to use colour maps for intensity coding. BB, A and Stern Special colour maps are available. It is also possible to draw a pixel cut graph using the *dist* button. A black and white or colour JPEG or GIF image can also be loaded, as well as a FITS[24] image

#### 3.2.1.2 Zooming

Aladin allows zooming of images by factors from 1/64 to 64. This can be done in two modes, either using the *zoom* button and choosing on the image display where to centre the zoom window, or by using the zoom panel which displays a thumbnail of the current image and allows choice of the zoom factor. The appropriate zoom window is then displayed on top of the thumbnail.

#### 3.2.1.3 Image comparison

As the VO allows access to data at several wavelengths at the same time, it is really important to provide image comparison/composition functionalities. In this category Aladin provides image contours, RGB composition, multiview, image blinking.

**Isocontours**

Using the *cont* button it is possible to create contours. The number and level values of the contours can be modified. Smoothing and noise reduction are optional. These contours can be overlaid to any other kind of image at the same position in the sky.

**Colour composition and image subtraction**

It is possible to create a coloured view by pressing the *rgb* button, and by assigning images at different wavelengths to two or three of the R,G, B channels. The pixel mapping of the three colours can be controlled independently. Some simple nearest neighbour resampling is applied when necessary. The same panel can also drive image subtraction.

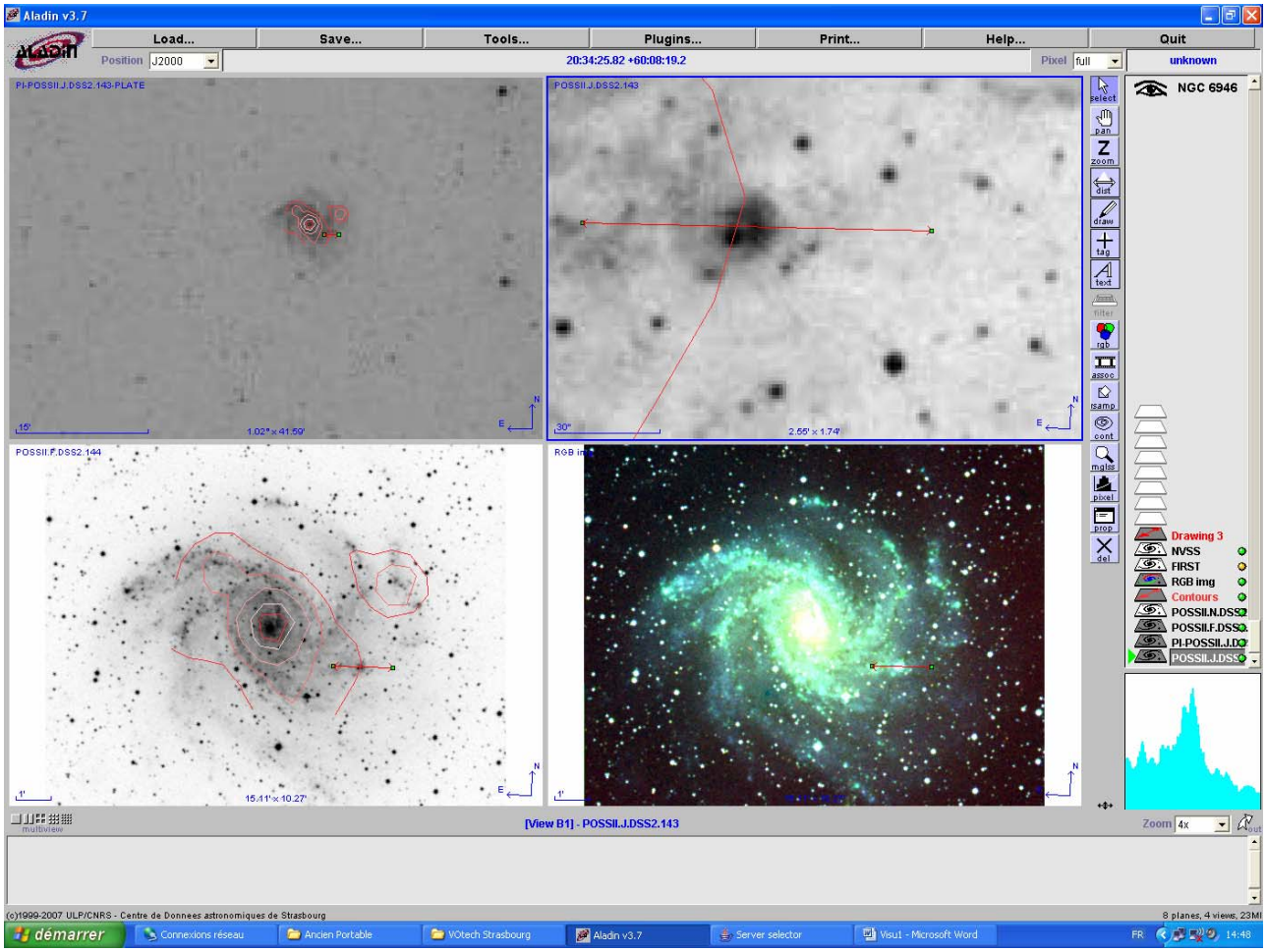


**Figure 2: Aladin in multiview mode**

**Multiview**

From version 3, it has been possible to use Aladin in multiview mode which allows the comparison of 2, 4, 9 or 16 images displayed at the same time. Small icons on the lower left part of the main panel allow control of the number of displayed images. Overlays allow association of regions and sources across all the images. Figure 2 shows Aladin in multiview mode; RGB composition of images in I, R and B bands, Schmidt plate global preview and overlay of NVSS[25] radio images.

## 3.2.2 Advanced functionalities

### 3.2.2.1 Blinking

The *assoc* button allows the creation of a new image plane from a sequence of currently loaded images in which these images are displayed in blinking mode. It is possible to accelerate, slow down, or stop the blinking

#### 3.2.2.2 Resampling

The *resamp* button allows resampling of one or several of the loaded images using either nearest neighbour or the bilinear algorithm. It is possible to speed up the computation by using 8 bits/pixel instead of full dynamical range of the data.

#### 3.2.2.3 ROI extraction

It is possible to extract small images around a set of preselected positional overlays (e.g. from a catalogue) in an image by using "Create ROI views for the selected object" in the menu displayed when clicking on the right button of the mouse.

#### 3.2.2.4 Cube visualisation

The next version of Aladin extends image functionalities to data cubes with two spatial dimensions. After loading the full dataset, which is necessary for the estimation of the autocuts, the cube is "displayed" in blinking mode, one channel after the other, with all Aladin blinking facilities available. It is possible to extract one or several channels in separate image planes. The *tag* button allows selection of a pixel in the current plane and the display of the 1D "spectrum" at that position.

## 3.3 Visualisation of Tabular Data in TOPCAT

### 3.3.1 Introduction

TOPCAT is an integrated interactive graphical tool for visualisation, analysis and manipulation of tabular data. In practice, it is most often used for source catalogues, and several of the application's features are specific to the needs of this kind of data, but it can be used for other astronomical or non-astronomical point data sets too. This section describes the aims and design of the program as regards visualisation capabilities and gives a (non-exhaustive) tour of the visualisation features.

### 3.3.2 Design

The main aims of the tool are as follows:

**Scalability**
Source catalogues can be extremely large, in some cases far too large to be able to investigate interactively using a GUI tool. However, the aim was to extend its capabilities as far as possible in this direction, so that artificial limitations on dataset size would be minimised. There are many aspects to this requirement, including ensuring that dataset size is not limited by the amount of available memory, using plotting, matching and analysis algorithms which scale well in memory and CPU usage, and providing display options which make visual sense even in the presence of many plotted points (e.g. more than the number of pixels on which to display them). TOPCAT is generally advertised to work well on datasets up to a million or so rows and tens or hundreds of columns, even using relatively modest hardware.

**Interactivity**
Catalogue data typically represent samples from a highly multidimensional parameter space - tables with hundreds of columns are not unusual. It is by no means easy to discover all of the interesting

patterns in the data under these circumstances. TOPCAT provides one way of addressing this issue by enabling the user to perform many different kinds of visualisation and other exploration of the data, building iteratively on patterns which become apparent from this exploration. For this purpose the plot→select→replot cycle must be (a) easy to set up and modify and (b) rapid to execute. Providing visualisation facilities which can be used in such an interactive way enables modes of data exploration which would be simply impractical for more `batch'-like models of data access and visualisation.

**Responsivity**
Fast operation is always desirable. The particular case for it in a tool of this kind is to encourage (or avoid discouraging) the exploratory analysis of the data mentioned in the previous point - if a plot or replot takes a couple of seconds it is easy to zoom in, zoom out, make point selections, see where they fall in other parameter spaces, etc, while if such operations take a couple of minutes the user is less likely to make the effort to examine data in speculative ways. In some cases (e.g. a scatter plot of a few million points) particular effort has to be expended to ensure that responsivity is maximised even where very large datasets are under investigation.

**Ease of use**
TOPCAT has quite a large number of features, many of which will not be relevant to any particular user. In practice, users do not spend much effort reading documentation, so the problem of advertising these features is a significant one - effort writing code can easily be wasted if nobody ever knows it is there. The design of the TOPCAT GUI is intended to make it as clear as possible what features are available without placing much burden of active enquiry on the user. Thus as far as possible all the functions of each window are available from two sets of immediately visible controls, viz. (a) a number of input widgets (selectors, checkboxes, sliders, text fields) labelled to indicate what the user is supposed to do to make the window work; and (b) a number of toolbar buttons with icons suggestive of their function (though this is in some cases rather optimistic - there is only so much you can do with 24x24 pixel icons).

**Comprehensive documentation**
Though experience shows that few users will read much of it, full documentation is provided. The manual is in two parts, an explanatory description of how the program works, and a reference part with a section for each of the program's (many) windows. The relevant reference page can be viewed easily from the window it documents, and the whole manual is available in HTML or PDF form.

**Ease of installation**
One of the hardest aspects of writing successful astronomical software is getting astronomers to use it. People are much more likely to try out (and hence have the chance to find out if it is worth their while to use) a tool if it is easy, and preferably *very* easy, to install and get working. Happily, the advent of Java makes it straightforward to provide a single downloadable file which will run on all important platforms, avoiding the various OS and library compatibility issues which applied to traditional compiled applications. With this in mind, the temptation to incorporate platform-specific non-Java *native* libraries has been resisted (with some minor exceptions for distinctly optional program features).

**Scriptability**
Interactive exploration of data is useful, but does not fulfil all the needs of most astronomy applications. In most cases an exploratory phase which allows the astronomer to understand (at least some aspects of) the data is followed by a production phase in which a repetitive, and perhaps

complex, task has to be invoked on many similar datasets. Ideally then, it should be able to automate the tasks which can be performed within the GUI application for hands-off use, rather than having to perform the same point-and-click sequences for many datasets. This function is performed with respect to TOPCAT by the STILTS [26] package, which forms its command-line counterpart, providing most of the same facilities. Although the two packages use much of the same infrastructure, the integration between them is not currently as good as it could be; having worked out a sequence of steps in TOPCAT, it requires some additional expertise with the software to set the same thing up in STILTS.

## 3.3.3Visualisation Features

TOPCAT provides a number of windows for graphical display of data. Section 3.3.3.1 describes the features which are common to these windows, and 3.3.3.2 gives a tour of the different windows available.

### 3.3.3.1 Plot Window Common Features

For ease of use and maintainability, as far as possible the different visualisation windows have a common interface and way of doing things. The features shared amongst them all are described in the following subsections.

**Plotting Styles**

For each kind of plot, different data sets (and subsets, or row selections, thereof) can be identified using different plotting styles. For the various scatter plots these can be configured individually for each subset in point of colour, shape, size, transparency and whether points are joined by lines or not. The other kinds of plots have similar options available as appropriate. As a convenience, it is possible to reset the style of all the displayed subsets as a group (for instance, show all points as single pixels, using a different colour for each set).

The ability to configure transparency of data points is of particular importance for visualising very large data sets. As the number of points plotted approaches and exceeds the number of pixels on the plotting surface, it is clear that some points are bound to get plotted on top of each other, which can obscure information. By allowing the user to configure plotting styles of variable transparency, it becomes possible to distinguish pixels with many points plotted on them (darker shade) from those with fewer (whiter shade).

**Linked Views**

Multiple plots can be made from the same data by using different plot windows. There is no limit to how many windows a single dataset can participate in. Within each window, different subsets of a single dataset (table) can be identified graphically by using distinct plotting symbols as described in Section 3.3.3.2 for each one. These subsets can be defined in many ways, including algebraically (by supplying a boolean expression based on column values), graphically (by dragging out a region on a plot) and visually (by clicking on rows of the displayed table cells).

The same list of subsets is available in all the views of the table, so, for instance, one can do a colour-magnitude plot, define a subset of the points in the plotted dataset as all those within a graphically identified region (dragged out using the mouse), and then see where those points show up in a different plot, say of sky position in spherical coordinates.

There are also separate facilities for highlighting single points in different plots. In general if you click on a point in one of the plots, a marker will appear over it to indicate which one you have

clicked on. If the same row is represented by a plotted point in any of the other currently visible views, the same marker symbol will appear over the corresponding point in them. The same applies to the row in the table cell display.

**Overplotting**
Each plotting window provides a set of axes bounding a surface on which data can be displayed. The minimal useful configuration is to display a single dataset from a single table. However, in each case it is possible to display multiple datasets on the same plotting surface. Each plotted set is one of the currently defined row subsets from one of the currently loaded tables. One or more subsets from each of one or more tables can be plotted on the same surface. The symbols used for each subset are individually configurable as described in Section 3.3.3.2 and it is possible to control which appear plotted ``on top" and which ``underneath".

**Axis Selection**
Each of the plot windows requires you to select one or more quantities to plot - in the case of a 2-d scatter plot, these are the X and Y coordinates. Each axis is selected using a drop-down selection box giving the names of all the eligible (which usually means numeric) column names, so that this is easy to do. However, it is also possible to enter algebraic expressions here, for instance arithmetic combinations of column names. The expression language used for this purpose is powerful and, for the enthusiastic user, extensible. There are also checkboxes which cause the data to be plotted on logarithmic and/or reversed axes.

**Range Selection**
By default the axis ranges in each plot will be set to include all the points in the dataset(s) being displayed. This auto-ranging can subsequently be overridden in various ways. For 2-dimensional plots the user can zoom in or out using the mouse to focus on a particular visible region. The mouse can also be used to zoom in or out on a single axis where appropriate. It is also possible to type in some or all numeric axis bounds using the keyboard. Finally, there is a toolbar button which undoes all zooming and returns to the auto-ranged behaviour, which can be useful if you lose track of where you have zoomed to.

**Graphics Export**
Each of the plot windows allows you to export the plot that you have prepared in either GIF or Encapsulated PostScript format. The files thus exported are of publication quality. Though it does satisfy the needs of some users for publication-quality plotting, TOPCAT does not attempt to be a fully-functional end-to-end publication-quality graphics package; there are no facilities for font selection, tick mark placement, plot annotation or other essentially decorative modifications of the plot. Users which require this kind of control over the appearance of graphics are encouraged to use TOPCAT to locate the region of data which they wish to visualise, then export that data in some format understood by a more fully-featured plotting package for additional tweaking.

## 3.3.3.2 Tour of Plot Windows

A number of different visualisation windows are provided: broadly speaking they are all scatter plots or histograms in one, two or three dimensions. The figures show these visualisation windows in action.

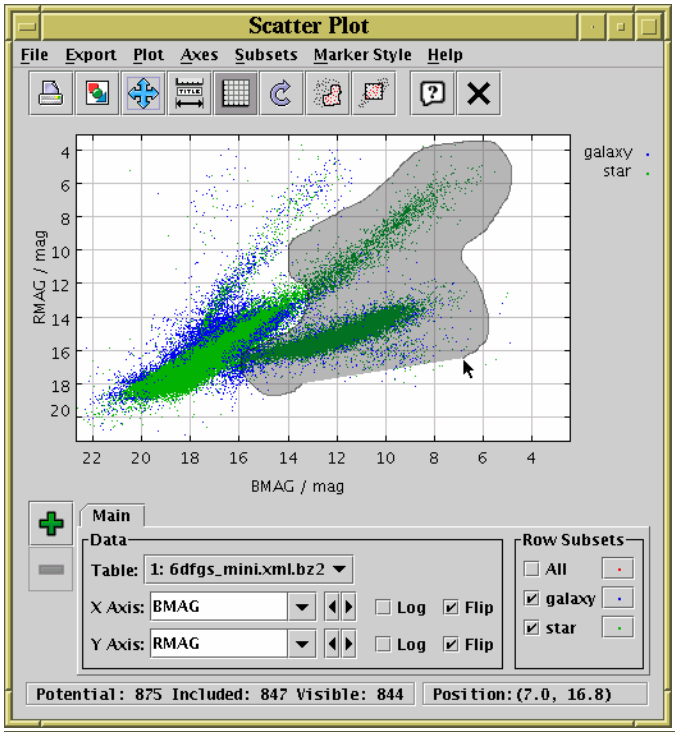


**Figure 3: TOPCAT Scatter Plot window**

**2D Scatter Plot**. In this figure star and galaxy subsets of a single data set are plotted using different colours. The user is in the middle of dragging out a region of the magnitude-magnitude parameter space of interest; this will be turned into a new subset which can be plotted in a different colour in this window or other windows displaying the same table.

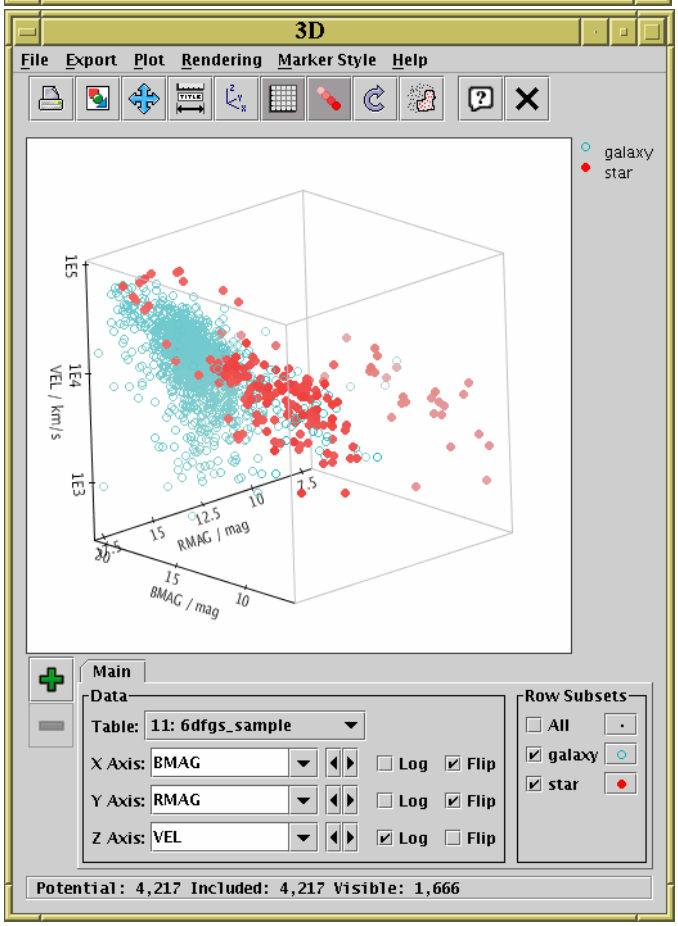


**Figure 4: TOPCAT 3D Scatter Plot window**

**3D Scatter Plot**. Here two axes are flipped so that they increase right-to-left and the third is scaled logarithmically. By dragging the mouse the user can rotate the axes around the centre of the displayed cube. The fogging effect which makes more distant points appear greyed-out can be turned on or off.

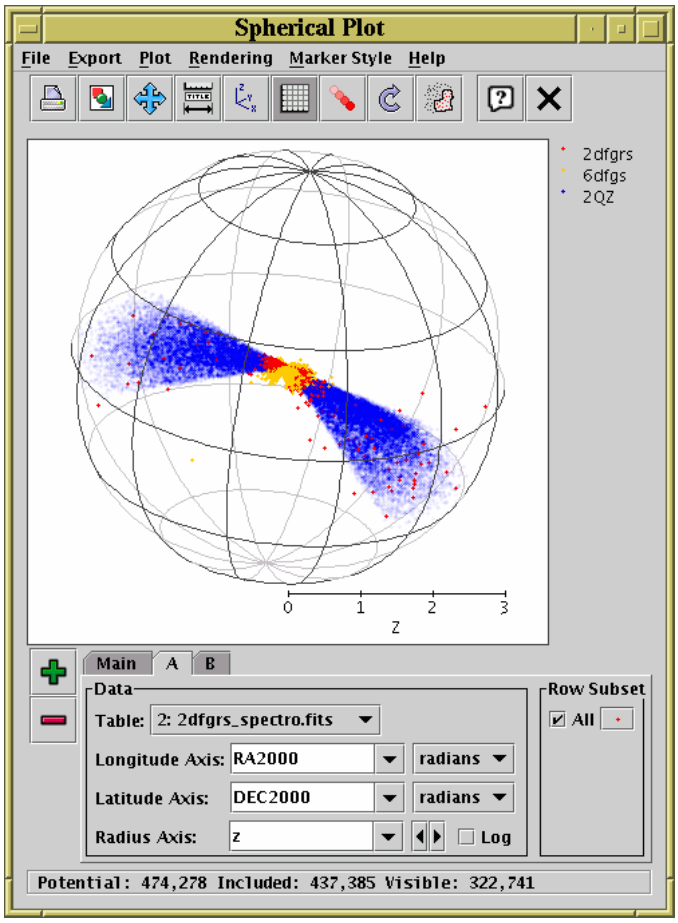


**Figure 5: TOPCAT Spherical Plot window**

**Spherical Scatter Plot**.
This is typically, but not necessarily, used to display the position of objects on the celestial sphere, with an optional radial coordinate (here redshift). The different colours represent different datasets (here, QSO positions from different surveys). Note the transparency of some of the plotted points makes it possible to identify more and less densely populated parts of the plot. Using the mouse the user can rotate the sphere and zoom in to the centre.

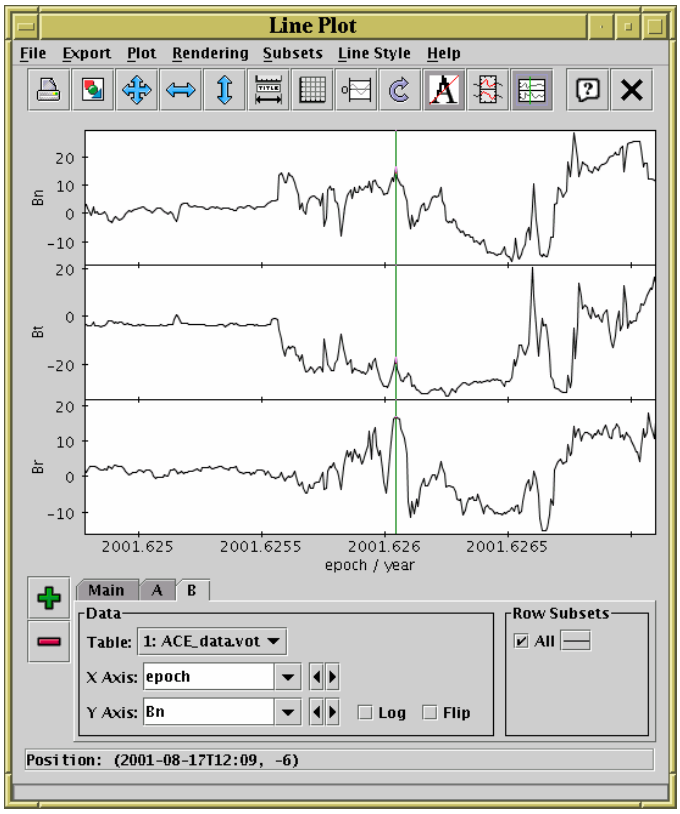


**Figure 6: TOPCAT Line Plot window**

**Stacked Lines Plot.** This plot, unlike the others, allows display of different data sets using the same X axis but different Y axes. Here it is being used to see the behaviour of three different variables as a function of epoch. Each Y axis (and the X axis) can be zoomed in or out independently.

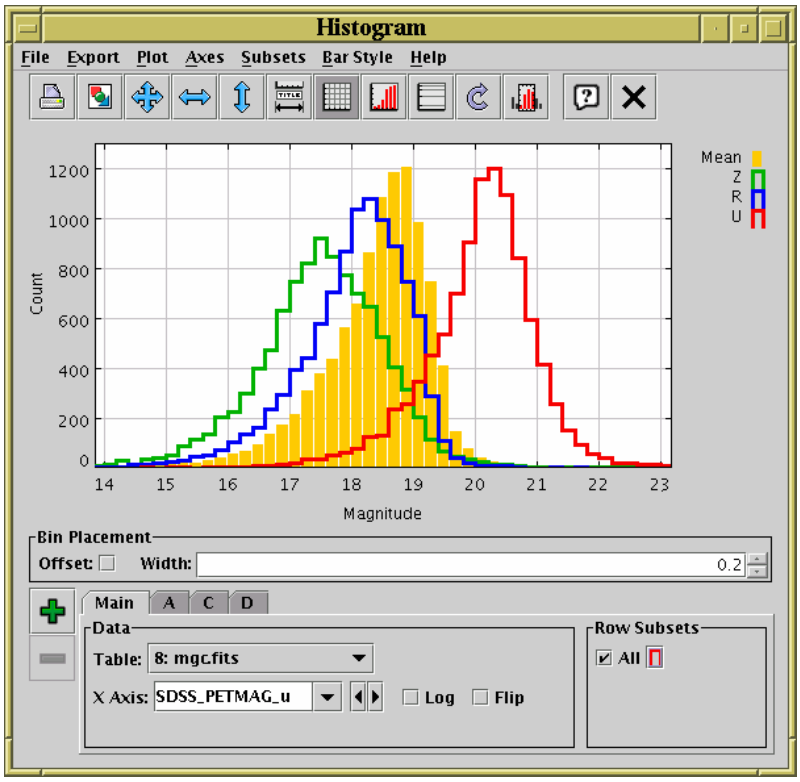


**Figure 7: TOPCAT Histogram window**

**1D Histogram.** Histograms of one or more single variables can be plotted using various visual styles. X and Y axes can be independently zoomed, bar width can be configured, and display can optionally be in cumulative mode.

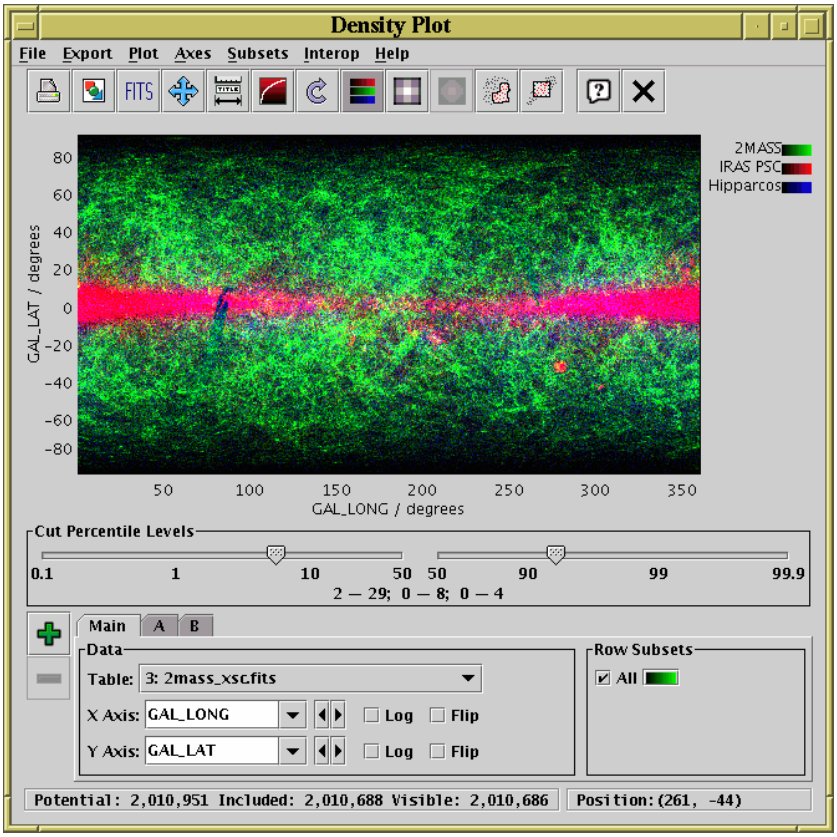


**Figure 8: TOPCAT Density Plot window**

**Density Map.** The density map is essentially a 2D histogram, in which the colour of each pixel is determined by the number of rows in the relevant dataset which fall within its bounds. Here three independent density maps are superposed using RGB colour planes. Sliders provide a crude way to determine low (black) and high threshold values for each colour channel. This provides a similar facility to the use of partially transparent pixels in scatter plots, allowing one to distinguish between low and high density parts of a crowded plane. Since the plot represents a grid, the contents of each cell being a number of counts, there is a facility to export it as a FITS array, which can be imported into an image viewing program (by saving it to file or directly using PLASTIC) to perform any required image analysis such as contour plots, cluster identification etc.

#### 3.3.3.3 Statistical Features

Some basic statistical functionality is present in TOPCAT. A window is provided to calculate simple univariate statistics (mean, variance, extrema, skew, cardinality, count of all/good/bad values, etc) for complete or subsetted datasets. Linear regression between two quantities can also be calculated and plotted. More sophisticated statistical features are not however provided at present.

#### 3.3.3.4 Other Features

TOPCAT provides a number of features not directly related to visualisation, for instance crossmatching, metadata display and modification, column calculations and manipulation, table concatenation, format conversion etc. These are not discussed further here.

# 4. DS6 Science Drivers

The scientific motivation of the VO rests on two related changes in the way that observational astronomy is done, namely the increasing importance of multiwavelength astronomy and the renaissance in survey astronomy. Taken together, these two necessitate a significant change in the way that astronomers work, as they shift from analysing small quantities of data on their desktop machines to working with large quantities of distributed data.

The VO is intended to provide the environment within which this new style of research is undertaken, so it must support the kinds of analysis which astronomers wish to perform with the new multiwavelength sky survey datasets. To a large extent these analyses are statistical in nature – whether measuring the spatial clustering of millions of galaxies, or identifying clusters within that spatial distribution, deriving summaries of the properties of populations of sources or searching for the one-in-a-million unusual objects, like high-redshift quasars or very low mass stars, which are outliers from that general population.

In some cases such analyses will require only the data from one sky survey, but in many others the science will be enhanced by access to other data resources. For example, in the case of quasar selection, as discussed in Section 2.2.1 above, the probabilistic classifier could be enhanced by information from radio or X-ray datasets.

Perhaps the most intriguing opportunities provided by the VO's data federation will be the discoveries which only become possible when new combinations of data are brought together, from different regions of the spectrum. These cases will require data exploration in the simplest sense, of exploring a data set to see what can be learnt from it. Such exploratory work is most likely to be interactive in nature, but may be followed by an automated phase in which the strength of a correlation spotted through interactive exploration of a subset of data is quantified by running a statistical test on the full data volume.

We believe, therefore, that the VO must support both interactive and non-interactive modes of data exploration, and that these are likely to require different software: interactive analysis implies client-side tools running on the desktop, whereas the automated analysis of large quantities of data is more suited to asynchronous jobs running at the site where the data already reside.

# 5. DS6: The VOTECH Data Exploration Design Study

## 5.1 Introduction

The DS6 workplan has been designed with the science drivers briefly outlined in the previous section borne in mind. Another important factor in planning DS6 is the finite resource available, which necessitates the re-use of existing tools – from astronomy and beyond - where the necessary functionality is available, albeit possibly re-using them in different ways.

In this Section we report the work conducted to date by the DS6 team. Most effort has been devoted to the client-side tools which will be used for interactive analyses: in Section 5.3 we describe the development of a protocol for making them interoperable, while in Section 5.4 we report on the results of, and lessons learnt from, VO-enabling a suite of such tools. The scalability challenges posed by the largest astronomical datasets – and those of highest dimensionality – are discussed in Section 5.5, and this is where much of the effort for the remainder of the project will be devoted.

## 5.2 Survey of existing software

One of the intended deliverables from DS6 is a survey of existing software which could be used for VO data exploration. An initial survey was produced as part of the background research leading to the design of Eirik (see Section 5.5.3 below) and can be viewed on the VOTECH wiki page [27].

A fuller version of this survey will be included in the final *Study Report*, but we note a few of the conclusions from that survey here. There is a vast array of data mining and visualization software available, but little of it is readily usable for VO data exploration, for two reasons. Firstly, many existing tools cannot handle data sets of the size or dimensionality frequently encountered in astronomy, while, secondly, the tendency is for each tool to have its own input data format, so, at the very least, translators must be provided to turn astronomical data (in FITS or VOTable or whatever) into the format required by each tool. In some cases – such as with Weka (Section 5.4.2) – where the tool is open source, it may be possible to amend the data input routines to handle astronomical formats directly, but this will only be straightforward if, as in the case of Weka, the tool's code is particularly well designed.

## 5.3 Interoperability of client-side tools

### 5.3.1 Introduction

PLASTIC is a protocol for communication between desktop astronomy applications. It is simple for application developers to adopt and is easily extended. Through PLASTIC applications can share data, link views or instruct each other to load an image of a particular area of sky. Although these operations are quite simple, this collaboration opens powerful new ways to exploit existing functionality. The motivation for PLASTIC is that the astronomer should have a suite of interoperating, specialized tools at his disposal which can be composed according to his particular needs.

## 5.3.2 Background

The capabilities of astronomy desktop applications are being continually enhanced with new features. While this is generally of benefit to astronomers there are limitations to this “bigger is better” approach. Inevitably, there will be some overlap between the applications' feature sets, as users demand capabilities from other applications in their own favourite, leading to a duplication of effort in the astronomical software community. As an application becomes more powerful its resource footprint usually expands, and its increased complexity may make it more difficult to maintain and navigate, even if a user only uses a fraction of the application's features.

The alternative and complementary approach is to encourage interoperability between applications, each a specialist in a particular task. This approach can be compared to the Unix philosophy of “Write programs that do one thing and do it well.”

At its simplest, interoperability is the ability of applications to share data through common formats. However, greater benefits may be obtained if applications are able to share functionality as well as data. Not only does this enable data to be transferred seamlessly across applications, but allows powerful data exploration capabilities such as cross-application linked views.

Below we discuss work that has taken place in the VOTECH consortium to address interoperability between desktop applications through a communication protocol known as PLASTIC.

## 5.3.3 Previous Work

Data exploration using linked views of the data is not a new idea: for example, the Mirage [28] and xmdvtool [29] applications both support several different visualization methods allowing the user to explore data simultaneously in different ways. While Mirage and xmdv are closed applications, with the different views built in, CDS has taken different approach with Aladin. Aladin exposes a publicly documented interface ("VOApp") that makes it easy for third-parties to write plug-ins to expand or reuse its functionality. This capability has been exploited by VOIndia's VOPlot [30] and CADC's Octet [31] applications. The symmetry of the interface means that such plug-ins are peers of the Aladin application itself and are able to send as well as receive commands. However, plug-ins are restricted to being Java applications and must run in the same JVM[3]. Furthermore, Aladin only exposes a limited subset of its functionality through the interface. The Aladin and VisIVO [33] teams overcame the Java-only limitation by adapting VisIVO, a C++ application, to control Aladin through the latter's scripting interface. The coupling between the applications is much looser, and a greater proportion of Aladin's features are exposed, but the architecture is no longer symmetrical and Aladin is unable to access VisIVO in return.

At the 3rd SC4DEVO workshop [34] it was recognized that there was a need to generalise the VisIVO/Aladin interoperability work to arbitrary applications and that the technology developed for the Astro Runtime [35] offered a means to do it. The PLASTIC protocol that was conceived at this workshop has been developed by the VOTECH Consortium and has, at the time of writing, been incorporated into more than a dozen applications including TOPCAT (AstroGrid/Starlink), Aladin (CDS), VisIVO (INAF), VOSpec (ESA), AstroWeka (VOTech) and Reflex (SAMPO).

[3] For a comparison of the Aladin plug-in interface and the PLASTIC protocol described here see [32].

## 5.3.4 Technical Overview

There were three main requirements for the PLASTIC protocol: it should be platform- and language-neutral, extensible, and above all simple to learn and use. PLASTIC works through a locally running daemon application called a “PLASTIC Hub”. Applications communicate with the PLASTIC Hub using one of several transport protocols: different protocols are supported to make it easier to adopt PLASTIC by application developers. Language neutrality was achieved by employing XML-RPC [36] as one of the underlying transport protocols, as there are XML-RPC libraries for all major languages. Platform independence has been assured by implementing the reference “PLASTIC Hub” (see Section 5.3.6) in Java. In contrast to the Aladin scripting interface, which has a well-defined set of operations, PLASTIC is based on messaging, thus allowing looser coupling and extensibility: new messages can be defined and the protocol extended as previously unforeseen uses arise. With the exception of some core system messages, the definition of new messages is delegated to application developers rather than being controlled by the PLASTIC specification team. Finally, the protocol has been kept simple as it was important for application authors to adopt it quickly, so that work could concentrate on the more difficult task of defining precisely how applications should interoperate and what messages were required. PLASTIC is not intended to be a general-purpose messaging system, and does not include features such as guaranteed delivery or encryption.

## 5.3.5 PLASTIC in action

The following example illustrates how PLASTIC works in practice, with different applications exploiting each other's strengths to explore data from Digby et al [37].

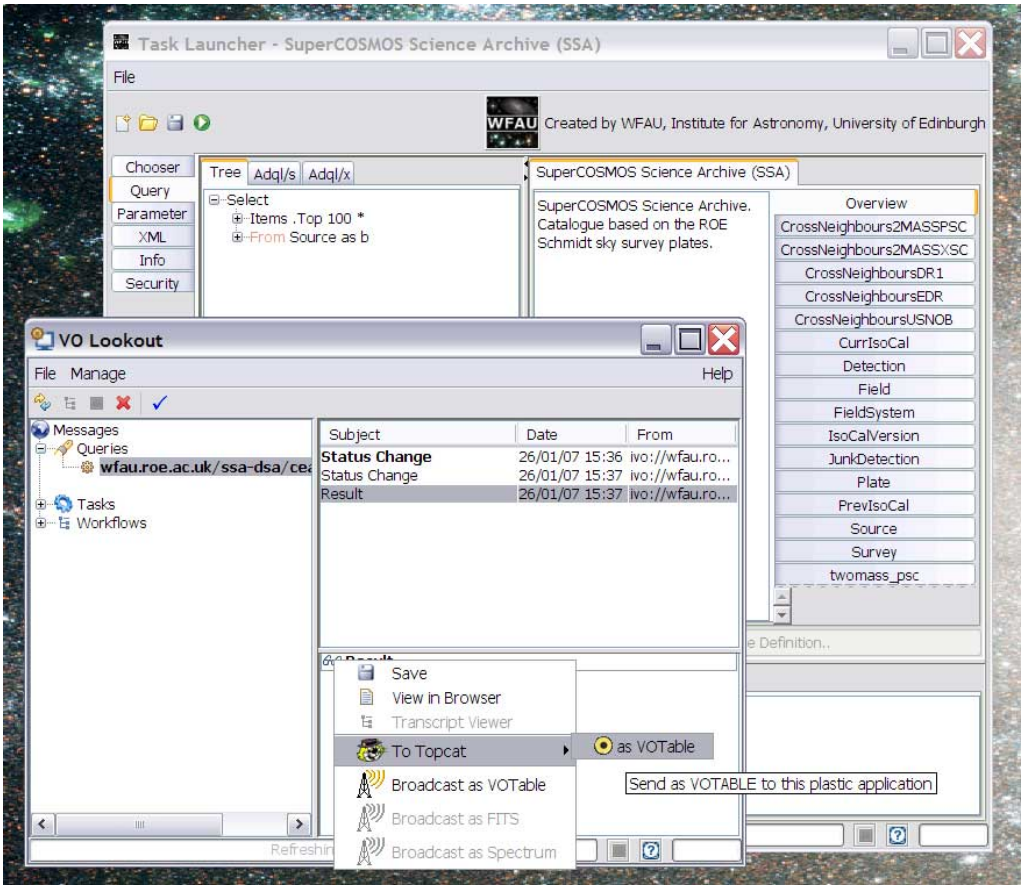


**Figure 9: Sending a VOTable to TOPCAT with PLASTIC**

The AstroGrid Workbench is first used to search for suitable data in the SuperCOSMOS Science Archive [38] and Sloan Digital Sky Survey. The astronomer begins his analysis by starting TOPCAT which locates the PLASTIC Hub running inside the Workbench and registers with it. The Workbench is notified by the Hub that TOPCAT has registered, and updates its menus accordingly, allowing the user to send the SuperCOSMOS data directly to TOPCAT (Figure 9).

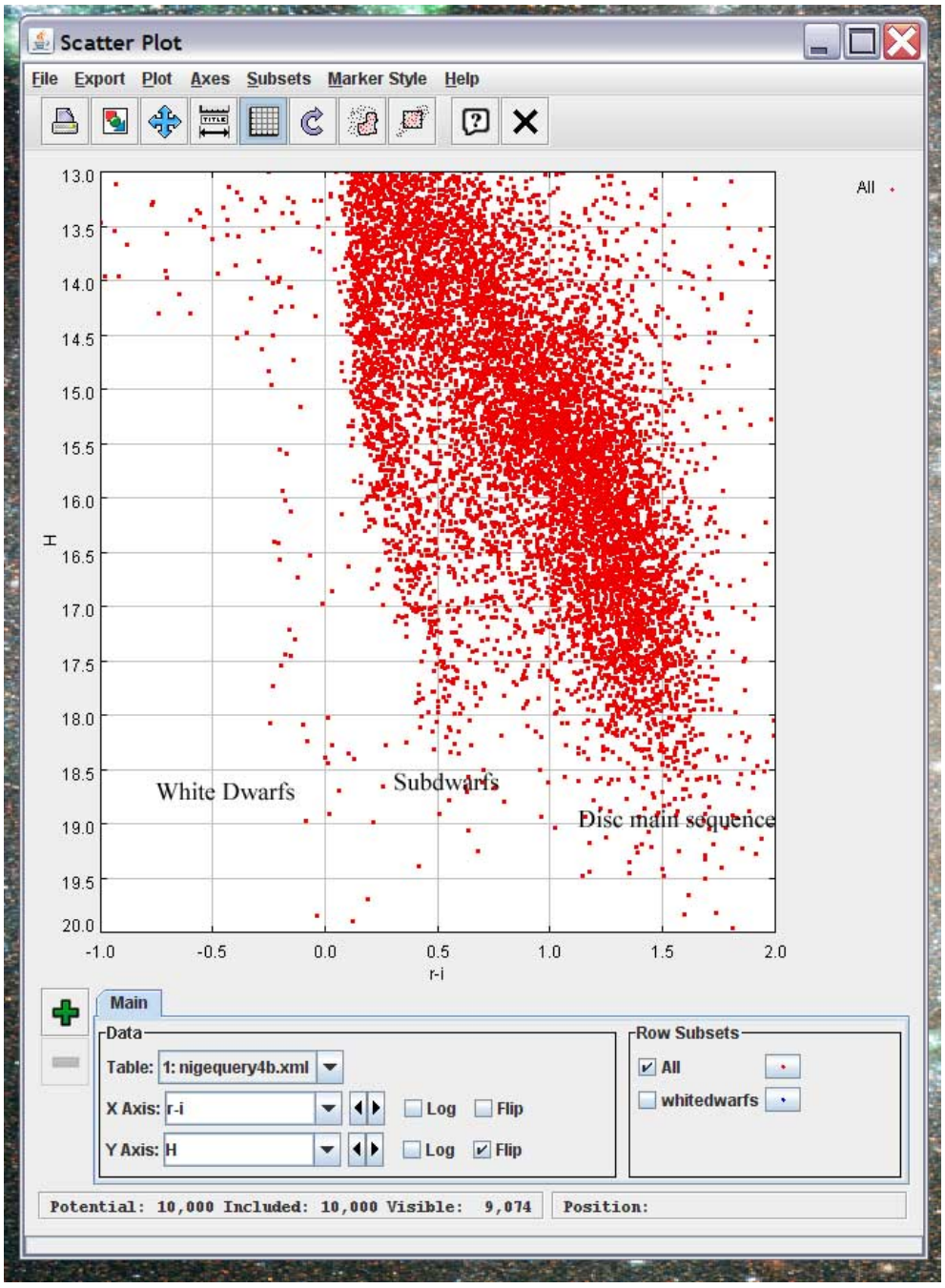


**Figure 10: Reduced Proper Motion Diagram**

The SuperCOSMOS data do not contain the attributes that the astronomer needs, so he uses TOPCAT to add synthetic columns to calculate the colour indices of the sources and their reduced proper motions. A scatterplot of the r-i colour index against the reduced proper motion reveals that the sources fall into three populatations: white dwarfs, sub dwarfs and main sequence stars (Figure 10).

For further analysis the astronomer starts VisIVO, a C++ application that specializes in plotting multidimensional data. VisIVO is registered with the PLASTIC Hub inside the Workbench, and shortly after the Workbench and TOPCAT's menus are updated to reflect the fact that VisIVO is running and able to receive VOTables (Figure 11).

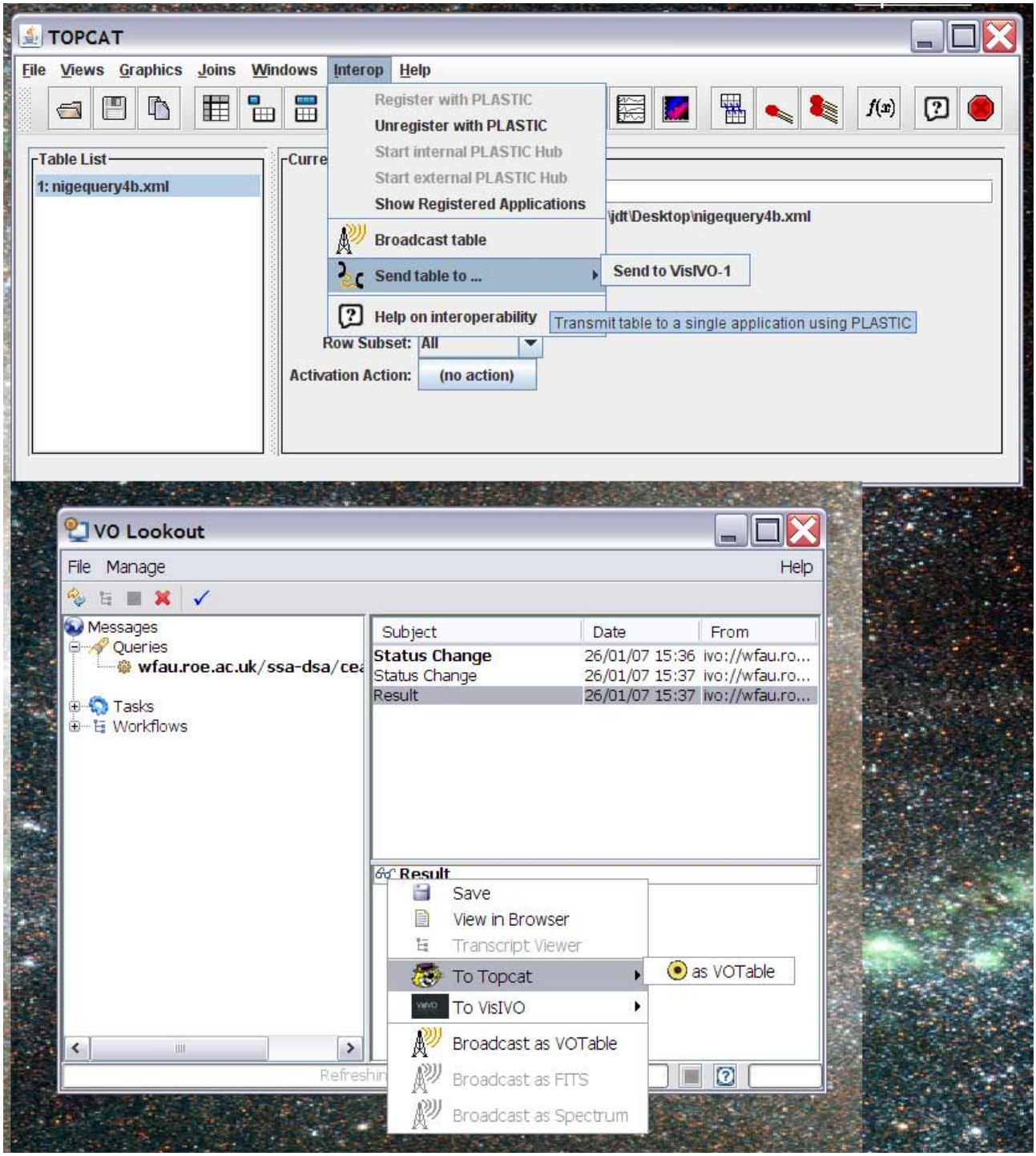


**Figure 11: Updating menus once VisIVO has registered with the Hub**

The astronomer sends the augmented data from TOPCAT to VisIVO and uses VisIVO to create a 4D plot of r-i colour index, g-r colour index, reduced proper motion, and magnitude (using the latter to choose the colour of the points from a lookup table) (Figure 12).

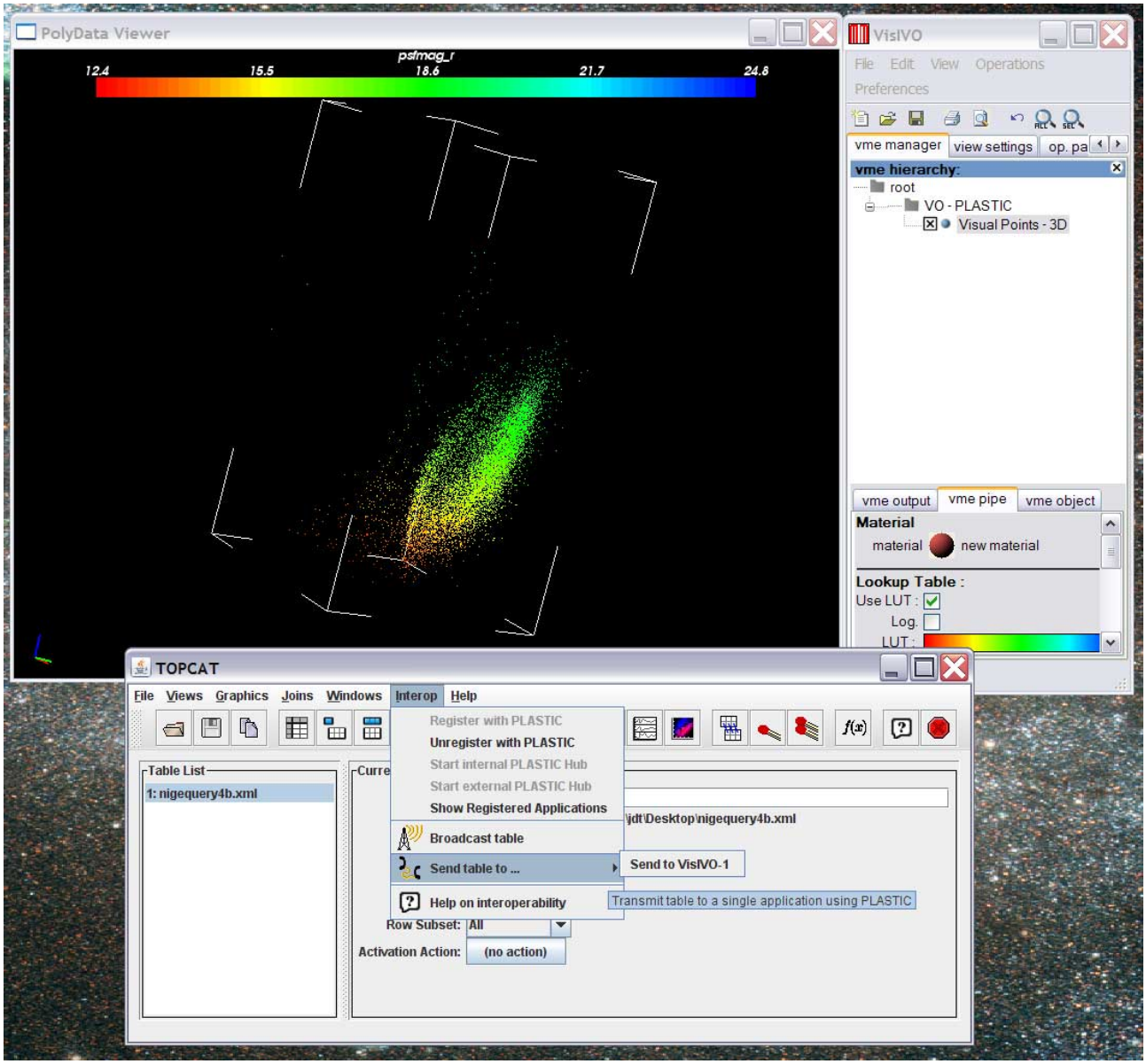


**Figure 12: Visualizing the 4D plot in VisIVO**

The white dwarf population can then be selected in TOPCAT and highlighted in VisIVO for further exploration (Figure 13). This workflow is, of course, greatly simplified. In reality the data would be transferred back and forth between the different applications, with interesting subsets extracted and outliers removed. It could be sent to statistical applications such as AstroWeka (see Section 5.4.2) and Eirik to aid the astronomer in identifying clusters and trends.

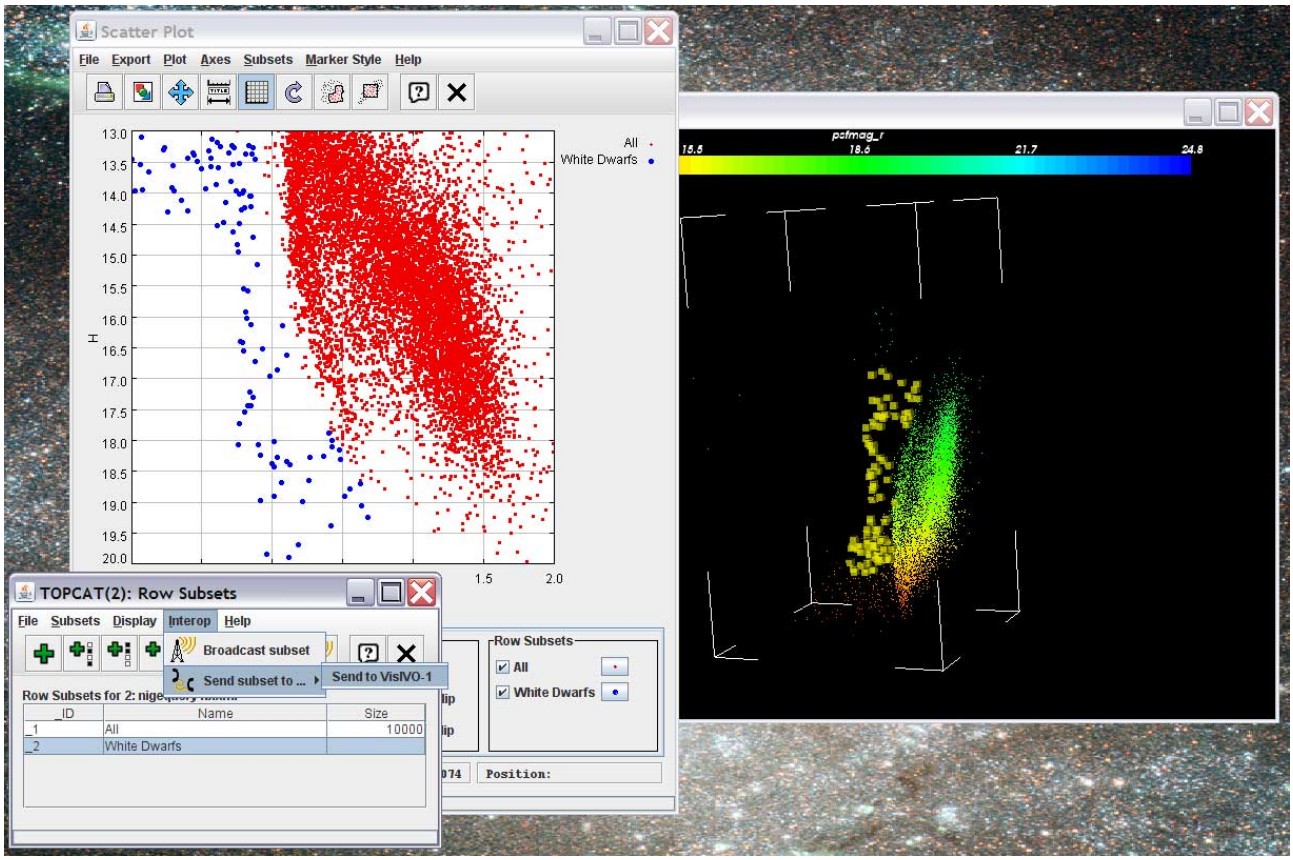


**Figure 13: Overplotting in VisIVO the white dwarf candidates selected in TOPCAT**

## 5.3.6 Technical Details

The following section gives an informal technical description of the PLASTIC protocol. Normative text can be found in Appendices A, B and C of the PLASTIC IVOA Note [39].

The two key concepts in PLASTIC are the PLASTIC Hub and PLASTIC Messages. PLASTIC is a specification for a simple messaging system: applications send each other messages requesting certain actions via the PLASTIC Hub, which routes them to their destinations. The advantage of this publish-subscribe architecture is that individual applications need only understand one of the Hub's transport protocols, and the Hub is responsible for any translation required. Furthermore, applications can discover other applications and their capabilities by interrogating the Hub. Note that many developers will choose to embed a PLASTIC Hub in their applications, and so the user will not necessarily be aware of the need to run an additional application.

The messages themselves consist of strings agreed by the application developers; some of them can be seen in Table 1 below. Each message may be accompanied by several arguments and may return a result and so is semantically similar to a remote procedure call. Again, the arguments and return values are determined by the application developers defining the message; they are not described by the PLASTIC specification and, with a small number of exceptions, the PLASTIC Hub has no understanding of their meaning.

The PLASTIC messaging system supports a number of different underlying communication protocols. As of the current version XML-RPC and Java-RMILite[86] are supported, but others

such as REST and CORBA could be added if there is demand.

XML-RPC is a language-neutral protocol and libraries are available for most major languages and PLASTIC functionality has been implemented within applications [87] written in Java, C++, Python, Tcl, JavaScript, Ruby and Perl. The Java-RMILite protocol makes PLASTIC particularly straightforward to use for Java programmers, but XML-RPC can be used instead if preferred.

It is important to note that PLASTIC is a specification, not an application, and anyone is free to implement a PLASTIC Hub. There are PLASTIC Hubs embedded in TOPCAT, Aladin and the Astro Runtime, the latter being the reference implementation, and work is currently under way to embed a PLASTIC Hub in Octet.

## 5.3.7 PLASTIC Messages

As stated previously, requests for an action are sent from one application to another by means of messages, which may be accompanied by a number of arguments and may also return a value. The ability to return a value means that PLASTIC is not a pure messaging system, but it simplifies short information requests such as "send me your name", and also allows clients such as scripting environments that cannot receive messages to obtain information from other applications.

Application developers are free to extend PLASTIC by defining new messages, but a set of core messages has been defined for system use.

PLASTIC message strings are arbitrary, opaque URIs. So far, PLASTIC application developers have elected to use URIs that are "IVORNS" (IVOA Identifiers[40]) as this allows their registration in IVOA-standard registries, and brings the following benefits:

- Guaranteed uniqueness
- A central reference where the meaning of a message can be stored
- Facilitating the automatic discovery of applications with particular capabilities

## 5.3.8 Example Messages

Tables 1 lists some illustrative PLASTIC Messages. A full description of the PLASTIC Messages currently in use is the subject of an IVOA note currently in preparation. All PLASTIC messages are optional, and applications can specify which messages they can understand and want to receive, if any.

| Message ID | Args | Returns | Description |
|---|---|---|---|
| `ivo://votech.org/votable/loadFromURL` | url: String,<br>id: String | success:<br>Boolean | Load a VOTable from the supplied URL. |
| `ivo://votech.org/info/getName` | void | name:<br>String | Get a human-readable name e.g. Topcat |
| `ivo://votech.org/info/getDescription` | void | name:<br>String | Get a human-readable description of the application. |
| `ivo://votech.org/votable/showObjects` | id: String ,<br>rows: int[] | success:<br>Boolean | Highlight/select the selected VOTable objects in the table with that id. |
| `ivo://votech.org/fits/image/loadFromURL` | url : String,<br>id: String | success:<br>Boolean | Load a FITS image from the supplied URL. |

**Table 1: example PLASTIC messages**

### 5.3.9 Current Status and Future Work

At the time of writing, the PLASTIC protocol has been widely adopted and added to more than a dozen applications, ranging from C++ and Tcl visualization tools, to a Java data-mining package. The future of PLASTIC is currently under discussion within the International Virtual Observatory Alliance, and it is expected to be adopted and developed by the new Applications Working Group.

## 5.4 VO-enabling existing data exploration software

In this Section we describe the process of, and lessons learnt from, VO-enabling a selection of data exploration tools. The reason for describing all of them in some detail is that they span a range of types of tool and the VO-enabling work was undertaken by people with a range of VO-related experience, as follows:

- **TOPCAT** was developed with one early VO standard (VOTable) in mind from the outset, but had to be later adapted to interface with others. This work was conducted by an experienced astronomical software developer, then working outside the VO community.
- **AstroWeka** is an adaptation of the Weka data mining toolkit for use in the VO. Weka was not designed for use in astronomy, and was a mature, existing package long before the VO was born. Most of this work was carried out by an MSc student with access to VO developers, but with minimal direct experience of the VO.
- **VisIVO** was intended from the outset to be a VO-compliant visualization tool. It was an extension of the earlier AstroMD[41] tool, developed by an experienced astronomical software development team new to the VO.
- **Aladin** was a mature (but ever-developing) software product before it was VO-enabled. Its authors have been at the centre of developments in the VO since its inception.

### 5.4.1 TOPCAT

TOPCAT is a graphical tool for analysis and manipulation of tabular data. It is not essentially Virtual Observatory based software - in many cases it will be run using data on local disk in an environment which has no need for network connections. Indeed the emphasis on scalability in its construction runs somewhat counter to the VO mantra of performing as much filtering of the data as possible at source and retrieving only small amounts for client-side processing.

However, much of the work done by astronomers now and in the future will clearly make use of VO standards and distributed ways of working, so where possible TOPCAT makes use of those standards to maximise its usefulness in a heterogeneous environment.

The following sections refer to different VO standards, describing how they are used and what was required for their implementation within TOPCAT. Some of these comments also apply to STILTS, TOPCAT's command-line counterpart.

#### 5.4.1.1 VOTable

TOPCAT was in fact originally conceived as a tool for viewing data in VOTable format. An early design decision however was to design it as a multi-format viewer, so that it deals with abstract

models of tabular data, which may have a number of possible serialisations on disk orelsewhere, one of them being VOTable.

There are always pros and cons of this kind of generic approach: the main advantage is wider applicability than being tied to a single format, and the main disadvantage is precluding access to some of the more specific features of each single format. TOPCAT's view of a table is influenced by VOTable (e.g. each column has a UCD) but not tied absolutely to it (the hierarchical structure of VOTables within a document and the GROUP elements which aggregate columns and parameters are not well represented).

The effect of this is that TOPCAT leaves something to be desired as a VOTable editing tool - loading a VOTable and then saving it can lose significant elements of the metadata. In view of the enhanced usefulness of the software for non-VOTable formats however (including FITS, CSV, ASCII, RDBMS and user-defined ones) the compromise was worth making.

#### 5.4.1.2 UCDs

Unified Content Descriptors (UCDs[42]) are machine-readable semantic metadata labels which can be attached to columns of a table to describe the kind of data which they contain. This is potentially of considerable use to a generic table analysis program such as TOPCAT, since it often needs to know the kind of data a column contains, or which column contains information of a given type. One example is that when performing a cross-match based on sky position, it is necessary to know which columns represent the celestial coordinates. In most cases the user will be asked to choose or confirm which column is to be used for what, but allowing the software to offer a default based on semantics encoded in the data can improve usability considerably (especially when the table contains several hundreds of columns).

TOPCAT propagates UCDs when reading and writing tables (where supported by the data formats in question) and provides some fairly crude facilities for selecting UCDs to attach to columns. It also makes some limited use of their semantic content, in particular to identify columns containing sky position and epoch coordinates. However, it is worth commenting that this exploits a very small part of the semantic richness provided by the various UCD vocabularies.

#### 5.4.1.3 Non-Local Filestores

AstroGrid's MySpace[43] and the emerging IVOA VOSpace[44] standard define ways of storing structured or unstructured data in remote filestores, to facilitate near-data processing and data sharing. TOPCAT provides facilities for direct access to these filestores in two ways.

Firstly, the interactive file browsers provided for loading and saving files provide exactly the same user interface when used with local filestores (local or network-mounted disk) or remote ones. A generic set of java classes which provides a GUI front end to an abstract model of a hierarchical file storage tree has been written, along with storage tree implementations representing the local filesystem and a remote MySpace storage facility. A tree implementation has also been written for the SRB (Storage Resource Broker[45]) remote storage facility, though this has not been much tested. The standard Java `javax.swing.filechooser` classes look like they ought to be able to provide this kind of abstracted model of a filesystem, but do not in fact employ sufficient generality to accommodate MySpace, so the code had to be written from scratch.

Secondly, URL-level access to MySpace is available using simple URLs such as ``myspace:/data/result.vot''. These have been implemented by using Java's pluggable protocol handler mechanism.

In both cases, the hard work of access to the remote data is done using the AstroGrid Runtime, leaving the TOPCAT-specific code to concentrate on user interface issues. Both of these components are available for reasonably straightforward re-use by other software which wishes to provide the same facilities. Recent versions of the AstroGrid workbench do use some of these components.

### 5.4.1.4 PLASTIC

PLASTIC is a protocol for transmission of data and control between loosely-coupled user applications. It provides the user with a way to view the results of processing with one tool in the context of another. In some ways therefore it eases the burden on the application developer, for instance TOPCAT does not need to provide sophisticated image viewing facilities itself, but can instead delegate such tasks to dedicated image viewers such as Aladin or GAIA by passing images to them as PLASTIC messages. Sharing views of data between applications in this way is inevitably less scalable and provides a less integrated user experience than if all functions were provided by a single monolithic application, but the advantages it confers in allowing multiple specialised existing applications to communicate represent a good compromise.

The functions which must be implemented to PLASTICise TOPCAT (or other applications) fall into three categories:

**Control**
The application must at least be able to locate and register with a running hub. In practice, it is also desirable to provide user-controlled ways to unregister or re-register, cope with hubs which disappear and re-appear during the application's lifetime, keep track of and keep the user informed about which other applications are currently registered and what messages each is capable of receiving, deal gracefully with erroneous responses from the hub and other applications, and so on. TOPCAT also provides the option of running a PLASTIC hub of its own, though this is only really intended for testing and experimental use.

**Message transmission**
TOPCAT is capable of sending various messages to other PLASTIC applications, transmitting whole tables, singles or lists of table row indices, sky coordinates (associated with a row) or images (density maps). When TOPCAT sends a table over PLASTIC it may be based on a previously loaded external table or on a synthetic one, for instance one resulting from a crossmatch. In either case the sent table will be a view of the table which may differ in significant ways from the original, for instance rows and columns may be permuted. This means that when transmitting, TOPCAT has to provide a new URL pointing to a table in that form (currently this references a temporary file which is deleted in due course) and must keep track of the relationship between the sent table and the current view of the corresponding table within TOPCAT; special care is required to ensure that incoming and outgoing messages referencing row numbers are handled correctly. For each type of transmission the user is given the option of sending to a single named application (selected from a list of currently registered applications known to receive the message in question) or to all listening applications.

**Message receipt**
TOPCAT can receive various messages from other registered applications. When a table is received it is loaded as if by user request from a file or URL. When a single row index relating to a known table is received, that row is ``activated", which means that the relevant point is highlighted in any existing graphical displays and an additional action may be invoked if so configured. When a list of row indices relating to a known table is received, a new ``row subset" corresponding to it is created, as if the user had specified it graphically or using an algebraic condition on column values. These latter two features effectively implement linked views of the same tabular dataset between different applications. TOPCAT also responds to various administrative messages which request information about the application name, description, logo, and so on.

Implementation was done in two parts: a standalone general purpose library providing utility functions, and use of this in the TOPCAT code.The general purpose library PlasKit[46] is available for third-party use.

#### 5.4.1.5 Remote Query Protocols

TOPCAT is able to make registry queries, and to query remote data servers using the IVOA SIAP (Simple Image Access[47]) and SSAP (Simple Spectral Access[48]) protocols. This functionality was introduced on a more or less experimental basis however, is fairly basic, and is not much advertised to the user. The reason for this is that TOPCAT is mainly a tool for analysis and manipulation of tables, rather than for interacting with the Virtual Observatory in all its forms. While it would have been technically feasible to add a slew of fully functioning VO-querying features it was judged that this would blur the aims of the tool and make it too unwieldy, even if just from the point of view of documentation.

Although neither AstroScope[49] nor PLASTIC were available at the time that this decision was made, those components neatly provide the functions of performing VO queries and transmitting the results to TOPCAT. There are therefore no plans to extend TOPCAT's inherent VO-quering functionality from its current rudimentary form.

### 5.4.2 AstroWeka

Weka is a popular open-source data mining package developed at the University of Waikato in New Zealand. It provides an extensive range of machine learning and data prepossessing tools, as well as a set of GUIs for operating them easily. Weka has been designed to be modular, with a common interface to each of its tools, allowing them to be combined to perform complex data mining tasks. The tools provided can be used to process and explore catalogue and image data, in order to find interesting objects. They can be used to help search for patterns and unusual objects, and can be trained to classify automatically new objects detected in the sky. In addition, clustering algorithms are available to split objects into similar groups.

Because so many different algorithms are provided in one package, it allows astronomers to try different techniques to find the one best suited to their requirements. When performing classification of objects, for example, the common interface provided allows an astronomer to set up an automated experiment which will try several classification algorithms, with different parameters, and evaluate which combination gives the best performance.

Weka is often used as a test bed for new learning algorithms, as its modular design makes developing these algorithms much easier. It takes care of housekeeping tasks, such as loading data from files and providing GUIs, allowing developers to concentrate on the more theoretical aspects of their work. Weka has a very active user/developer community, who constantly contribute new and improved machine learning algorithms for it.

In order to make Weka compatible with VO applications, it was necessary to adapt it to understand the VOTable format, communicate with AstroGrid services through the Astro Runtime (AR) and communicate with other client side tools through PLASTIC. Due to Weka's modular design, adding in new functionality was very straightforward. Many of the operations needed were provided by APIs such as STIL and the AR and only required a minimal amount of additional code to be written, gluing them together. AstroWeka, the VO enabled version of Weka is available from http://astroweka.sourceforge.net

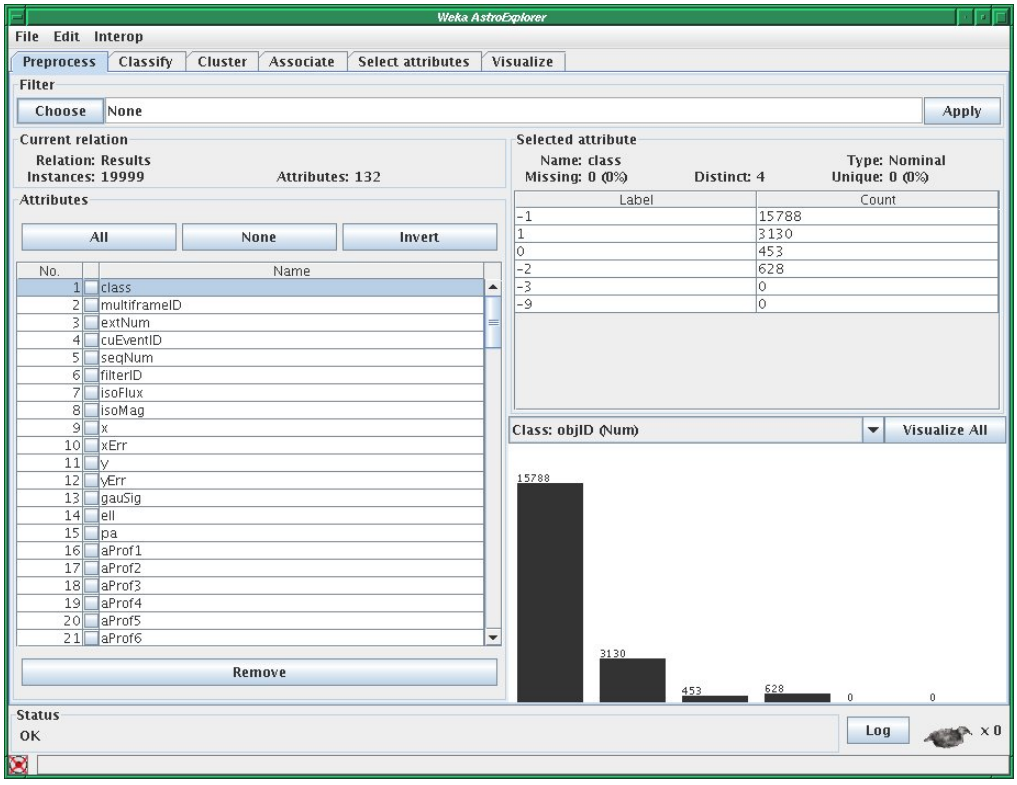


**Figure 14: The Explorer window in AstroWeka**

## 5.4.2.1 VOTables

Logically, Weka's native file format, ARFF, is quite similar to VOTable. Both consist of metadata and an associated table of data, though Weka's format is slightly more limited. Whereas VOTable documents may contain multiple independent tables of data, ARFF can only have one. Weka also has fewer data types to choose from, being limited to numerical, nominal, string and date types.

Allowing Weka to read and write VOTable format was the most labour-intensive part of the conversion. The main problem we faced was that Weka's native format, ARFF, has fewer data-types available than VOTable. For example, all numerical values must be converted to doubles which could potentially lead to round off errors. If the operations carried out in Weka are part of a

work flow involving other programs, this could cause further problems, as any integers in the data converted to doubles can no longer be tested for exact equality.

A further problem with converting data lies with the way it is stored; most surveys store categorical data as small integers, and do not provide any indication that this information is any different from normal integer data in the header of their VOTables. There are several ways of dealing with this, such as having the user manually specify how the attributes are to be treated, but they are not very satisfactory.

Once it had been decided how different data types would be converted, building the converter was relatively straightforward. Using the STIL library allowed easiy parsing of the VOTables, and once the relevant information was extracted, it could then be converted and loaded into Weka. Data are converted as follows:

- Attributes which have their optional values listed in the header are treated as categorical.
- Attributes whose specified type can be converted to doubles are treated as numerical. Any of these data which cannot be parsed are listed as missing.
- All other data are treated as strings.

### 5.4.2.2 Accessing Remote Data

Adding the capability to download from, and upload to, MySpace from within Weka's GUI was very easy, with most of the work being handled by the Astro Runtime. All that was really required was to add extra buttons to trigger the events. Similarly, we added the ability to search for data services that have a cone-search interface and then load data sets from them. Again, this was relatively easy, though it did require the development of some additional dialogs.

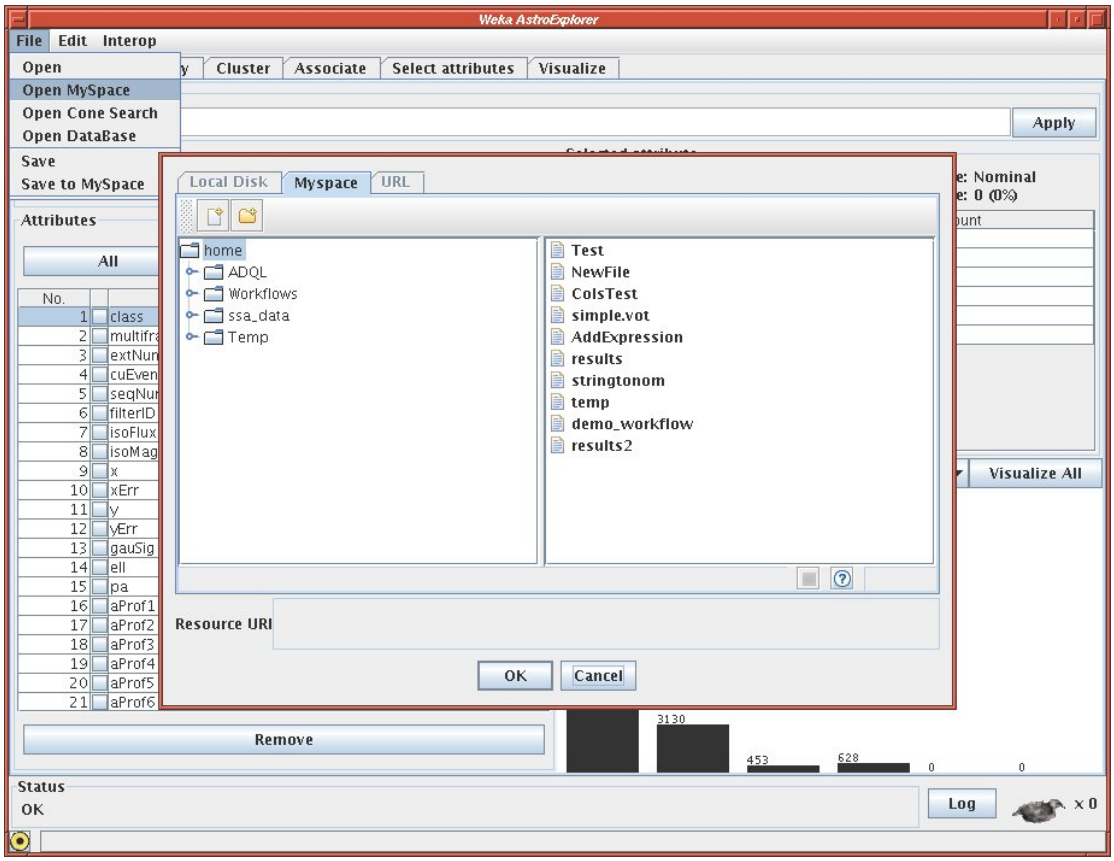


**Figure 15: Accessing MySpace from AstroWeka**

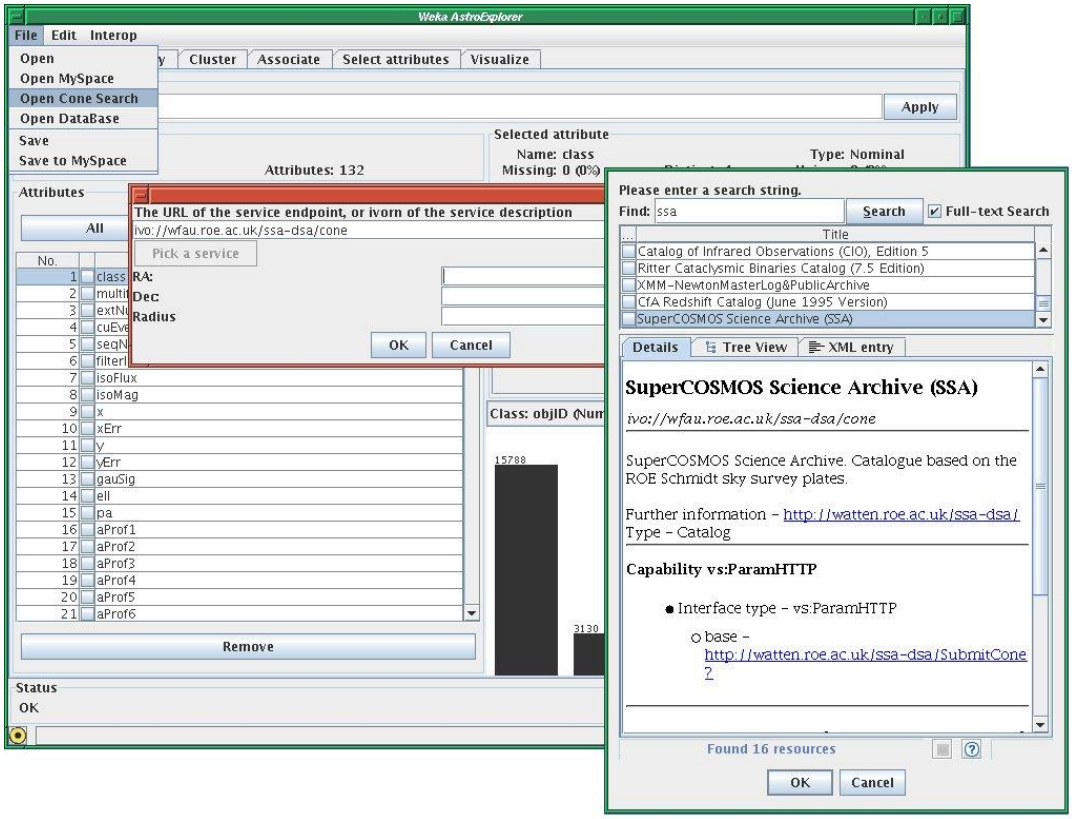


**Figure 16: Finding and using conesearch services with AstroWeka**

### 5.4.2.3 PLASTIC

In order to PLASCICise Weka we made use of the PLASTIC Java library. This library takes care of connecting to the PLASTIC hub and responding to the core messages, and makes writing actions for new messages very similar to working with the event queue in Java GUI programming.

One problem we encountered, was that Weka's data loading methods are written with the assumption that they will be called from the window event thread, and operate asynchronously, which caused some problems when they were called in response to a PLASTIC message. As they return immediately the calling application will assume that Weka has completed loading the table before it really has, and may delete temporary tables before Weka has a chance to load them. In order to remedy this, it was necessary to rewrite some of Weka's data loading methods.

### 5.4.2.4 Further Work

**Scalability**

One of the main criticisms leveled against Weka is its lack of scalability; that it is simply not as robust as some commercial packages. Many dismiss Weka out of hand as it is implemented in Java, which is as less efficient than C++. This particular criticism is unfair as modern Java Virtual Machines have become very efficient, and Java's automatic garbage collection means there is less likelihood of memory leaks occurring.

A more valid concern is Weka's dependence on virtual memory: it must load all data into memory and will crash if it runs out. While this does prevent it from being used on extremely large datasets directly, with the right precautions it can still of great use for performing initial exploratory analysis. We are trying to experimentally establish bounds on the size of data sets that Weka can be used on, to give astronomers a practical guide as to when Weka is useful: the results from this work will be included in the final *Design Study Report*.

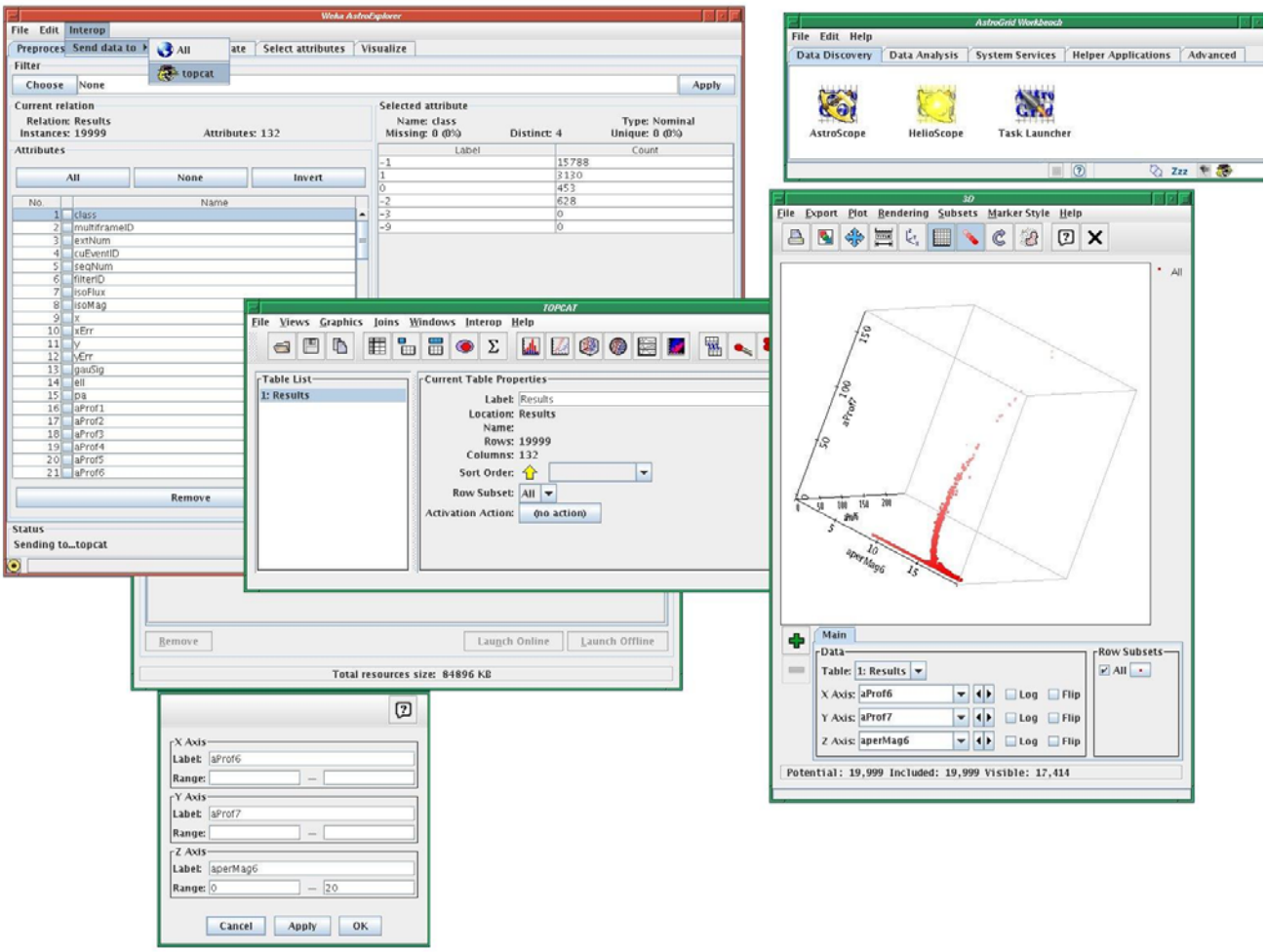


**Figure 17: Using AstroWeka and TOPCAT together, via PLASTIC**

In addition we are looking into adapting Weka to make it more scalable. Possible avenues include adapting it to use hard disk storage when necessary, similar to the STIL package, and integrating it more closely with database systems, removing the need to load all data into memory at once. Using secondary storage in this way will provides stability at the cost of computational time. New, more memory efficient algorithms, relying on data structures such as k-d trees and ball trees, are also being investigated.

### Additional GUI Functionality

Currently VO extensions are only available in the Explorer GUI, we are in the process of extending the Experimenter and KnowledgeFlow GUIs to also allow access to MySpace and other data services in AstroGrid.

## 5.4.3 VisIVO

### 5.4.3.1 Introduction

Data represent a critical resource for scientists, and in particular, for astronomers. Observational instruments (telescopes, satellites etc.) produce enormous quantities of images and information. Computers and numerical simulations generate huge amount of data and all these data must be stored, managed and analysed. These steps require a great human effort, large scale facilities and efficient, powerful tools.

In recent years the Astrophysical community has started a joint effort, in order to create an infrastructure which could provide the maximum availability of data and common tools to deal with them. This infrastructure is the Virtual Observatory. The development of the VO is coordinated by the International Virtual Observatory Alliance (IVOA). The IVOA work encompasses all the aspects related to data management, access and manipulation. Most of the work of IVOA has focused on observational data. However, the interest toward theoretical data, produced by numerical simulations, is rapidly growing. Broadly speaking, the goal of the Theoretical VO (TVO) is to create a database of simulated data, accessible from anywhere by web-based tools in a easy and transparent way. Using the same metaphor of the VO (and of the web), its just like as if the researcher has all the simulation data in his/her pc. Data mining, analysis and visualisation of these data require tools capable to interact with the TVO and data from catalogues, accessing and exploiting its resources.

VisIVO, the software that we present in this section, is one of these tools. It as an effective instrument for the visualization and analysis of astrophysical data. It is VO standards compliant and it supports the most important and popular astronomical data formats such as ASCII, FITS, HDF5 and VOTables. Data can be retrieved directly connecting to an available VO service (like, for example, Vizier[50] and loaded in the local computer memory where they can be further selected, visualized and manipulated. It can deal with both observational and simulated data and it focuses on multidimensional datasets (e.g. catalogues, computational meshes etc.). It is completely open source and the binaries and sources can be downloaded from the web site http://visivo.cineca.it.

At present the software is fully tested only for Windows XP platforms and the version for Linux is in progress. This work is only one of many different applications that the Italian astronomical community is developing in the VO framework. This effort is led and coordinated by the National Institute for Astrophysics (INAF[51]), in collaboration with other institutions like, CINECA[52], the largest Italian academic supercomputing centre.

### 5.4.3.2 VisIVO a tool for 3D Visualization

VisIVO is specifically designed to deal with multidimensional data. Catalogues or numerical simulations, rather than 2D images, represent the basic target of VisIVO. Different quantities can be visualised and treated at the same time. The architecture of VisIVO strictly reflects the structure of a typical scientific application built on the Multimod Application Framework[53]. On this framework VisIVO implements all the elements that are specific to the visualization and analysis of astronomical data. VisIVO embodies a “select object, apply operation” utilization metaphor which is somehow similar to that of many graphical commercial applications (such as Adobe PhotoShop). The usage of VisIVO generally begins with an operation of data loading. Data types that are

currently supported by the application are VOTables, FITS, HDF5 as well as ASCII and binary raw data (dump of memory). The result of a loading/importing/modifying any type of data is represented in a data tree as a node. In order to display data loaded in tree, it is necessary to instantiate a View and associate it with the chosen tree node. A View is a rendering window that gives a particular representation of the data tree. VisIVO currently supports the following Views: Points, Vectors, Volume Rendering, Isosurface, Glyph, Stereo, Histogram. Each view displays data according to a specialised visualization pipeline. It is possible to instantiate several Views and each View can display a different subset of the data tree. This is a point of strength of VisIVO as many specialised views, displaying different (or same) data, can be instantiated and data can be manipulated and analysed independently of the views displayed. The properties of the selected view (e.g. camera perspective, axes, logarithmic scale) can be adjusted via the contextual menu.

In order to analyse and modify data loaded in the tree, a set of operations are available to the user. There are operations that simply modify the data, operations that perform some statistical analyses on data and operations that do both things. All of these generate output nodes that can be displayed according to the type of output (e.g. An output that represents statistical 2D data can be displayed in a Histogram View).

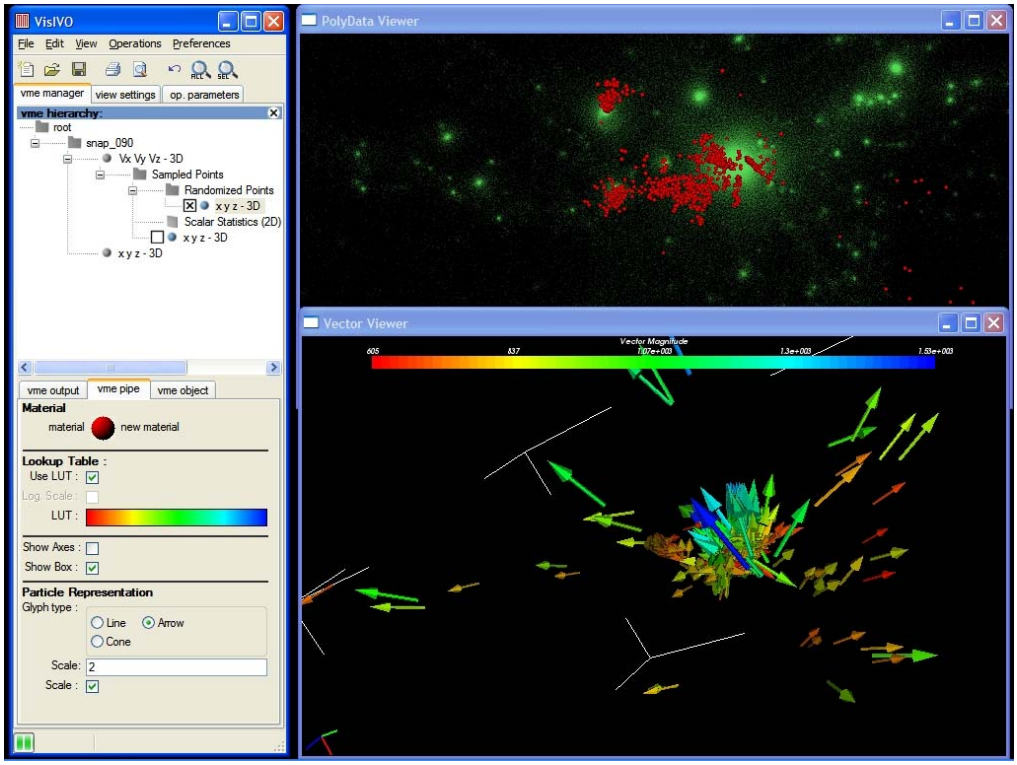


**Figure 18: VisIVO visualizing an HDF5 dataset**

VisIVO supports several visualization techniques for data. Unstructured points are visualized as pixels. The user can set the transparency of the pixels and their colour. It is possible to colour each pixel differently, according to an associated quantity. For example, gas particles can be coloured by their temperature (e.g. blue for cold particles, red for hot particles). Points can also be described with glyphs (solid geometrical shapes), whose shapes and size can be parametrized to some physical quantities associated with data. Figure 18 shows VisIVO visualizing an HDF5 dataset using the points viewer to show the points position and the vector viewer to show the velocity field associated with the points. Here Glyph visualization is used to highlight points of interest. On the left is VisIVO's control panel.

Mesh-based data can be visualized with two different techniques, isosurfaces and volume rendering. Isosurfaces are surfaces characterized by a given fixed value of the plotted quantity. They separate regions with higher values from regions with lower values. Different isosurfaces can be calculated and visualized at once. They can be characterized by different colours, according to the contour value. Using the volume rendering technique, different values of the quantity are represented by different colours and different transparencies. The overall effect is a cloud appearance. Figure 19 shows an example of isosurface view, where two isosurfaces are represented with two different materials.

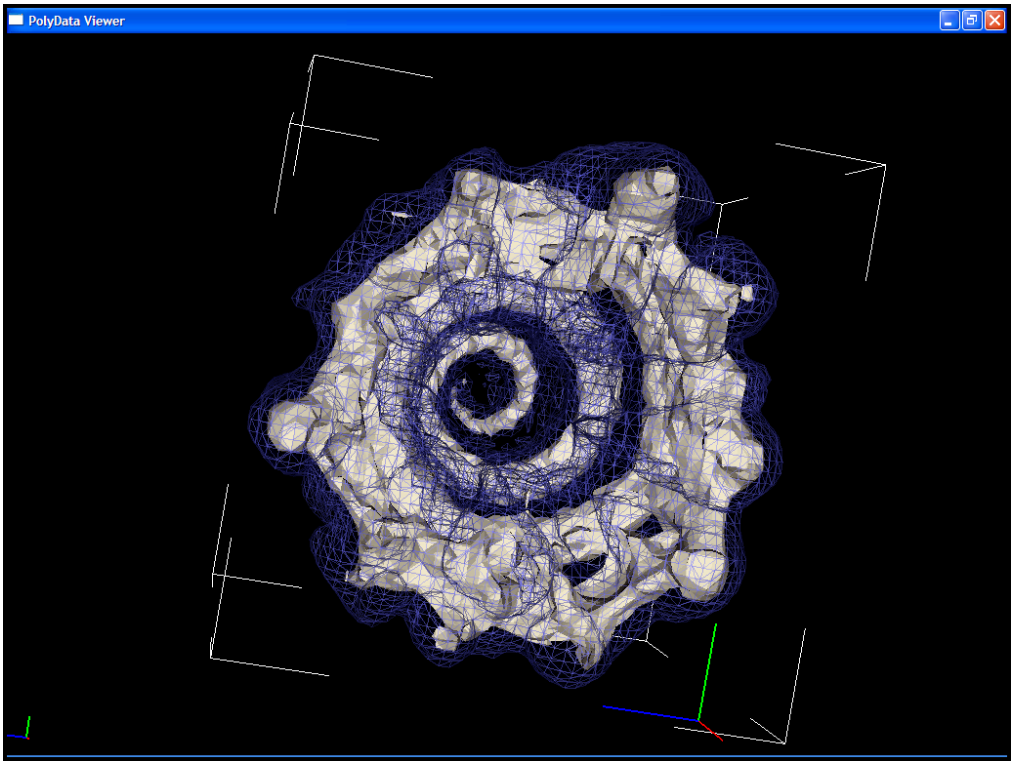


**Figure 19: An example of an isosurface view in VisIVO**

## 5.4.3.3 VO Interoperability

VisIVO is able to interact with many other VO-compliant tools using PLASTIC. VisIVO's interface to the PLASTIC hub allow the user to have many points of view of the data, not only the ones provided by VisIVO, the user can get images making VisIVO interoperate with Aladin, or directly see the loaded table data using TOPCAT,etc.

**PLASTIC-enabled tool**
The capabilities of VisIVO are extendable through PLASTIC. Equally, through PLASTIC, VisIVO's functionality is made available to other applications. Through PLASTIC, applications can share data and link their views of the data. Data exploration using disparate linked views of the data is not a new idea: for example, the Mirage and xmdvtool applications each support several visualization methods allowing the user to explore data simultaneously in different ways. However, PLASTIC enables linked views across applications.

The following example illustrates how VisIVO's PLASTIC interaction works in practice, with several applications exploiting each other's strengths. Suppose an astronomer wishes to analyse a dataset of astronomical sources for clusters in some subspace of the parameter space. He begins by loading the data into AstroWeka and this reveals several clusters that the astronomer would like to visualize. AstroWeka does not have any means for visualizing the data, so instead the astronomer executes VisIVO, and uses PLASTIC to send it the dataset from AstroWeka. The user selects the points that make one of the clusters in AstroWeka and this cluster is highlighted in VisIVO. This step my be iterated several times as the astronomer tries different clustering algorithms and attribute sets, with data being sent back and forth between AstroWeka and VisIVO.

Finally, the astronomer would like to visualize the clusters according to their spatial distribution overlaid on an image of the sky in Aladin. He sends the dataset from VisIVO to Aladin, loads an appropriate background image, selects a cluster in VisIVO, and the corresponding points are highlighted in Aladin.

While much of the above workflow could be accomplished by saving the dataset from one application and loading it in the next, PLASTIC makes the operation seamless. Furthermore, PLASTIC enables the cross-application linked views.

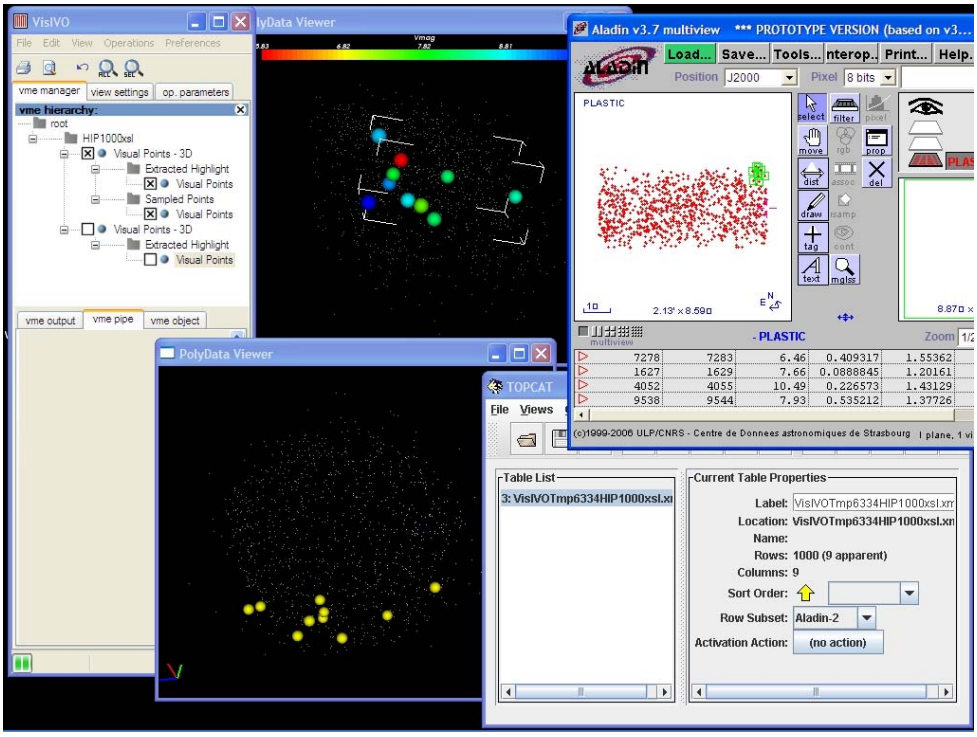


**Figure 20: An example of interoperability between VisIVO, Aladin and TOPCAT**

### VizieR Web Service data download

In its first phase of development, the only possible interaction of VisIVO with the VO was off-line. Data could be downloaded from usual on-line services in a standard format like VOTables or FITS and visualized by the software. However, now, VisIVO is able to query directly the VizieR web service to retrieve data from it and visualize them as if they were local data. This functionality allows the user to visualize and analyze remote data. The interaction with the service is transparent to the user. The user must only fill specified fields with the parameters defining the data he wants

to download. The result of this operation will be a list of catalogues, and selecting one of them, data will be visualized as it they were in a file.

### VWS VisIVO Web Services: an example of Grid capability

We have developed a Web Server prototype with Grid capabilities: a server side tool for VisIVO. VWS runs under Apache Tomcat 5.5.9 servlet container using Axis 1.2.The VisIVO server side runs in a Grid User Interface based on both INFN Grid 2.4.0 and gLite 1.1. We have used the GILDA User Interface Plug & Play. This software allows us to deploy grid enabled web services. We have chosen, as first Grid-enabled service, a group-finder software. A group finder can be a very time-consuming code: the CPU and memory usage increases with the number of particles. For large data sets the only way is to run remotely by using dedicated computing nodes. We have chosen HOP group finder to develop the VWS prototype.

### 5.4.3.4 Conclusion

VisIVO represents the first experience of an immersive Visualisation and Data Analysis Tool in astrophysics. It is one of the first visualization tools that operates in the VO framework and its development will follow the IVOA recommendations and issues.

## 5.4.4 Aladin: image display

We describe in this section the motivations behind making Aladin PLASTIC compatible, the actual implementation of PLASTIC in Aladin, and how PLASTIC has helped increase the ways that Aladin can be used, illustrated by various examples.

### 5.4.4.1 Pre-PLASTIC era

The PLASTIC protocol was born from the need to make desktop tools interoperable. In the pre-PLASTIC era, the Aladin team had already proposed several solutions - coming out from concrete use cases - to tackle this issue:

- The *VOApp interface* (previously known as *ExtApp*) is a Java interface defining possible interactions between Aladin and another Java application. This interface is currently used to interface Aladin with VOPlot (a 2D plotter developed by VO India) and with APT (the Astronomer Proposal Tool, developed by the STScI[54].
- In the frame of the AVO demos in 2004 and 2005, Aladin was interfaced with the spectra viewers VOSpec ([55],developed by ESAC) and Specview ([56],developed at STScI). This interface was developed through direct calls to ad-hoc Java methods.
- Aladin can be controlled by an external script or application by sending script commands via the standard input (STDIN). This method has been used by VisIVO, and has been now deprecated in favor of PLASTIC.

Each of those approaches has drawbacks or limitations:

1. The VOApp interface is limited to Java applications, running in the same Java Virtual Machine. It also means that interconnected applications have to be packaged together.
2. Calling directly works well for specific collaborations, though also limited to Java applications running in the same JVM. Moreover, this solution is not flexible, nor extensible.
3. The third solution, passing script commands through STDIN, does not allow callbacks from Aladin to the calling application. Thus, the communication is one-way only, an application taking control of Aladin. Aladin has no channel to communicate back to it.

PLASTIC overcomes those limitations, notably the programming language barrier, and provides a two-way communication channel. Moreover, PLASTIC's philosophy to let each application independent from each other, gives much more flexibility. In other words, the list of applications Aladin can collaborate with is not predefined. Those arguments led us to make Aladin compatible with PLASTIC, this protocol being a generic solution for the interoperability between astronomical desktop tools.

### 5.4.4.2 Making Aladin PLASTIC aware

A first prototype of a PLASTIC compatible Aladin could be developed in a few days, relying on the existing entry points and general architecture developed for the VOApp interface. This early prototype implementation had following benefits:

- Proof of concept: early PLASTIC adopters helped to demonstrate the capability of the PLASTIC concept.

- Implementation feedback: the prototype implementation helped to identify problems in the PLASTIC standard itself and to resolve them. It also helped to debug the reference PLASTIC hub, basis of the architecture.

In order not to disturb usual Aladin users with PLASTIC teething problems, it was decided to integrate PLASTIC-related developement in a prototype version freely available but separated fromthe official release. Thus, users interested in PLASTIC features in Aladin could test it and provide useful feedback.

Given that the PLASTIC protocol is stabilized, and that the PLASTIC implementation in Aladin is well tested and reliable, PLASTIC features will be fully integrated in Aladin version 4.

### 5.4.4.3 Solving usability issues

Allowing users to test an early implementation of PLASTIC in Aladin allowed identification of some bugs, but also revealed some usability issues. Indeed, it was found that some users had difficulties understanding the *PLASTIC hub* concept: the fact that one must launch an external application, the hub, then connect the different applications to the hub, before being able to make the different applications interact, was not intuitive for some "beta testers".

In the light of that, it was decided to integrate a hub within Aladin, and to launch it by default in case no other hub is currently running, and to also to connect to the hub at start-up. In this way,

users do not have to understand the internal working of the PLASTIC machinery, and can still use it in a more friendly manner. The integrated hub in Aladin is Mark Taylor's microhub, minimal version of PlasKit. It fit our needs, still being very light (less than 100KB). Advanced users can easily choose (from the PLASTIC preferences window, see Figure 21) not to launch the internal hub at start-up, thus allowing them to use any hub they want: the aim with installing an internal hub was to give the maximum of freedom to the user, though providing an easy default behaviour for beginners.

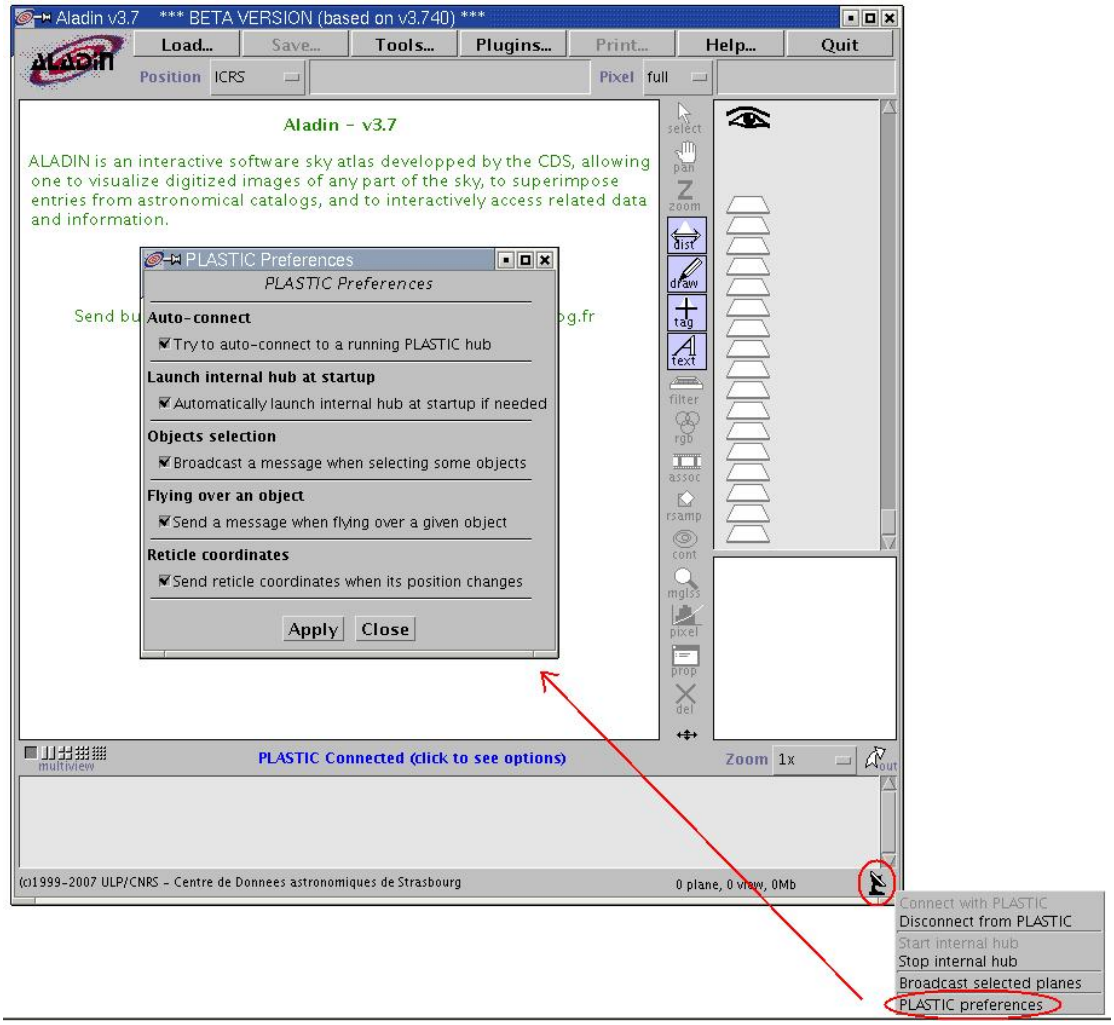


**Figure 21:The PLASTIC preferences window in Aladin**

Another improvement in term of usability was the addition of a small icon reflecting the current status of PLASTIC (see Figure 22). At a glance, the user knows if there is a running PLASTIC hub, if Aladin is connected to it, and if Aladin can transmit to other PLASTIC-aware applications. This icon also gives access to the different PLASTIC options.

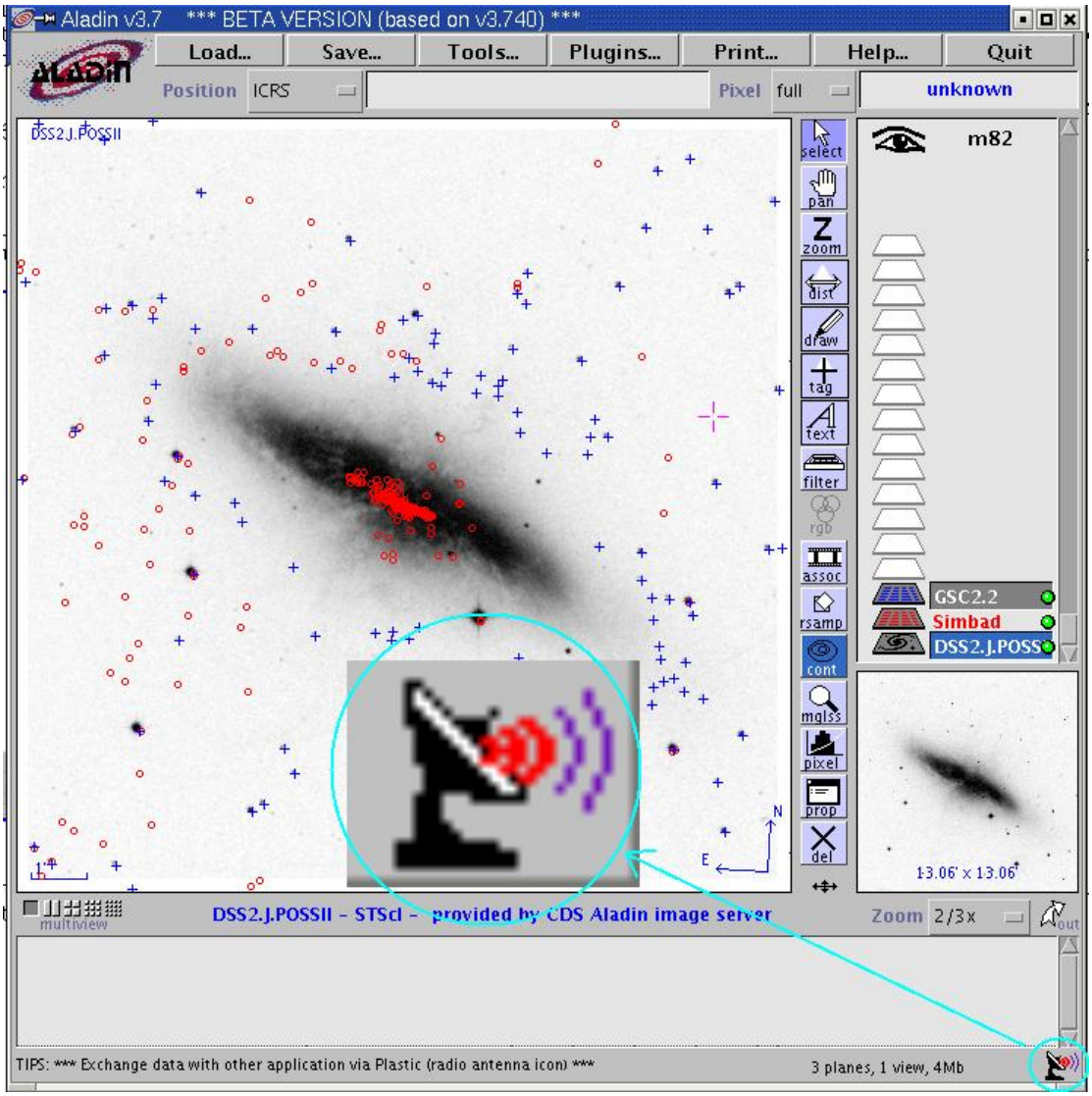


**Figure 22: The radio antenna icon in Aladin reflects PLASTIC status**

Figure 23 depicts the different possible states of this icon (from left to right):

- no hub is running
- a hub is running but Aladin is not connected to it
- Aladin is connected to the running hub
- Aladin is connected to the hub, and can interact with other connected applications
- Aladin is currently sending a message to the hub

**Figure 23: Possible states of the PLASTIC icon in Aladin**

An attempt was made to integrate PLASTIC features closely with existing Aladin UI elements. For instance, when a user wants to transmit an image or a table to another application, he just chooses the plane of his interest, right-click and choose *Broadcast* in the pop-up list (see Figure 24). In a similar way, data being displayed in a metadata tree (images, catalogues, spectra) can be visualized

in a PLASTIC-compatible application by right-clicking on and selecting the application to use (see Figure 25).

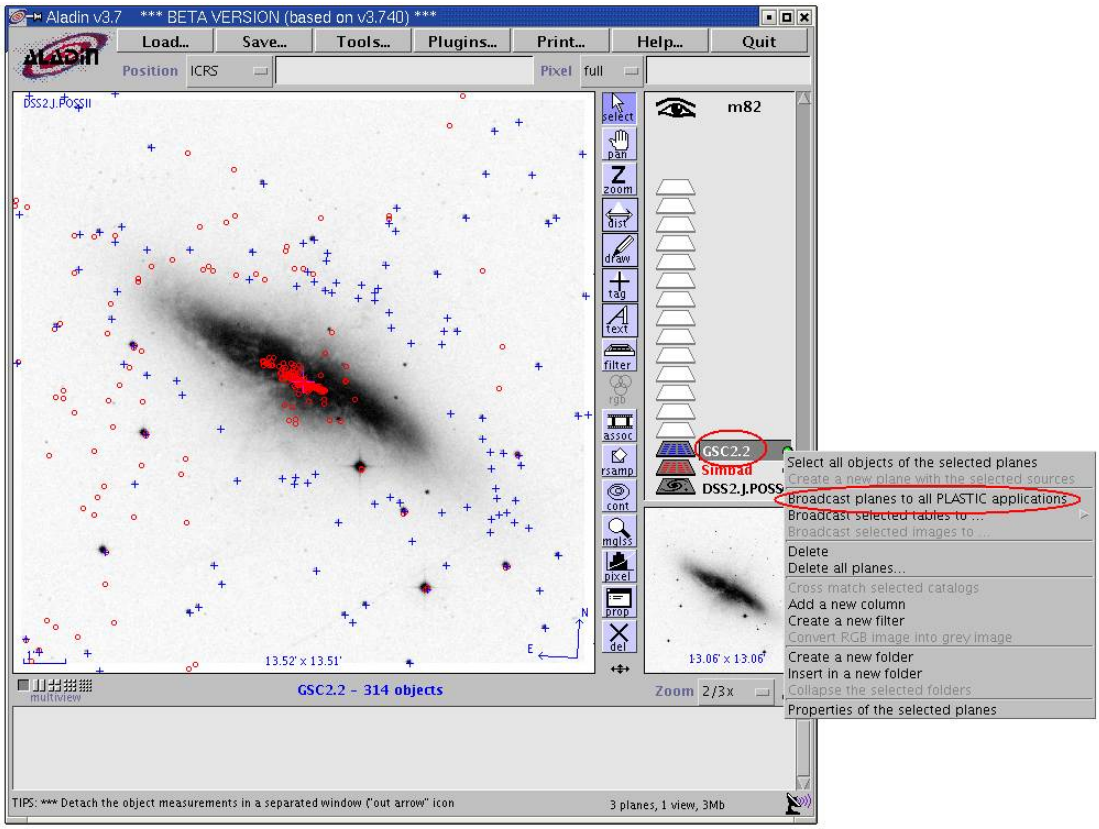


**Figure 24: Broadcasting an Aladin catalogue plane with PLASTIC**

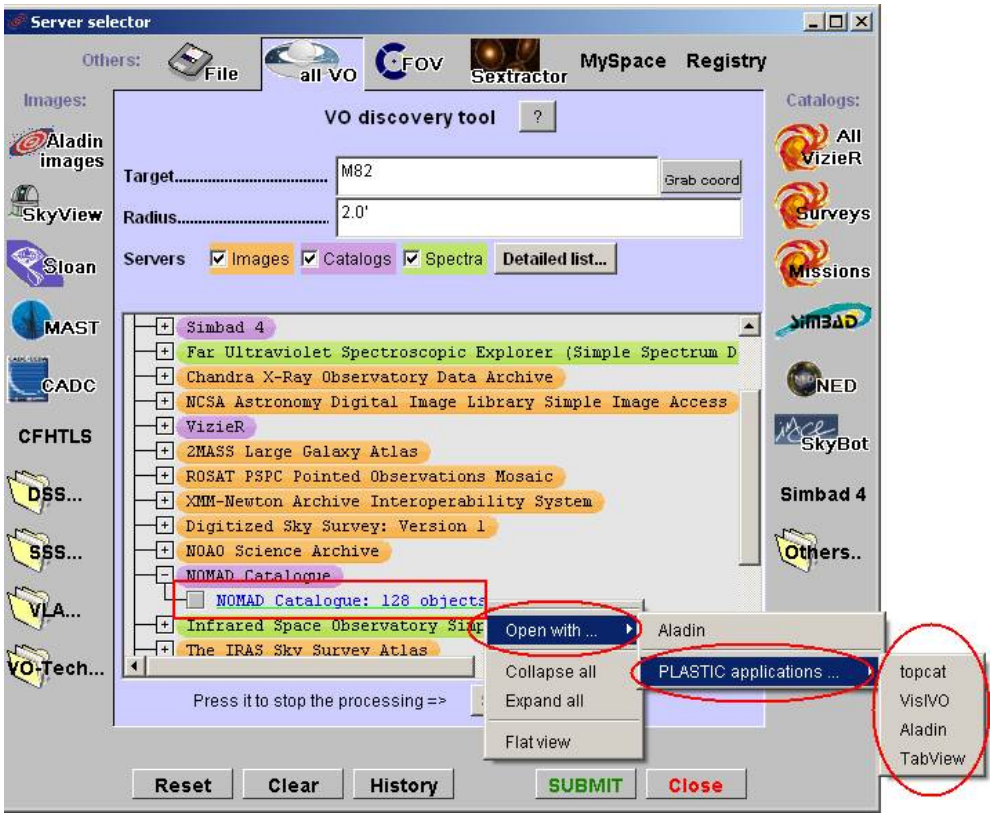


**Figure 25: Sending a catalogue URL from Aladin to a PLASTIC application**

### 5.4.4.4 Benefits

The PLASTIC compatibility has greatly widened the possible usage of Aladin. There is no more a limitation to the Java world. Moreover, possible interactions between Aladin and other applications are no longer dictated by the initial choice of developers, but are now dynamic and flexible. Not only can Aladin use other applications to extend its features, but the contrary is also true.

We will detail in the next paragraph what extra interactions are now possible using PLASTIC.

**Data Exchange**

Relying on PLASTIC and on existing format standards (FITS and VOTable), exchange of data (tables and images) between Aladin and other applications is straightforward and transparent for the user. He does not have to worry about saving data in one application, and reloading it in the other. His work flow is much smoother, as he can focus on his science, rather than having to fight with files and data formats.

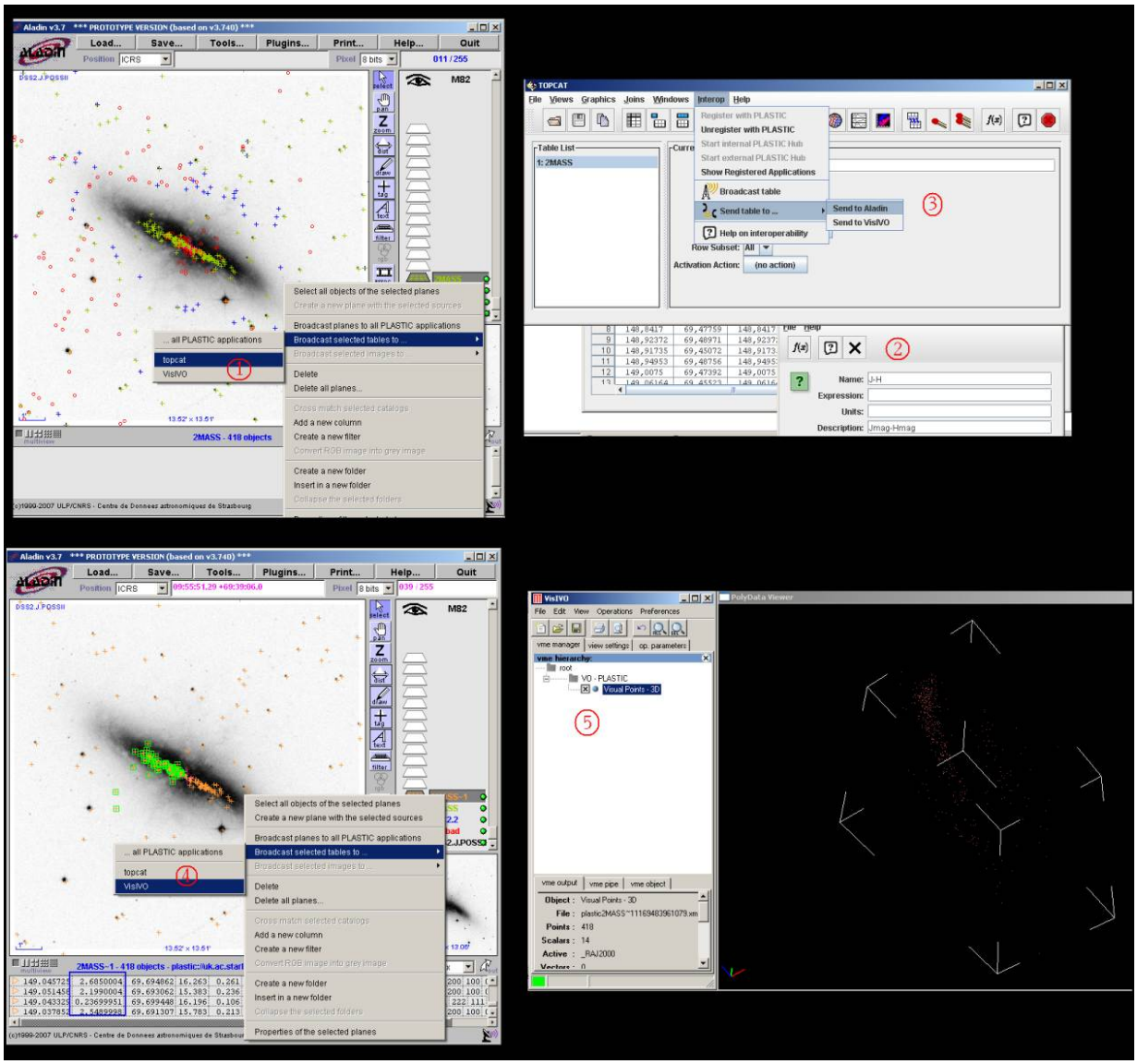


**Figure 26: Exchanging data between Aladin, TOPCAT and VisIVO**

For example, let us imagine a user loads a table in Aladin from a VO service and would like to add an additional column. He just has to send this table to TOPCAT, computes the new column, and sends the table back to Aladin. Then, he creates a filter to visualize the extra column, and how its value is distributed over the sky. If he needs a 3D plot, he can startup VisIVO, and send the table. Figure 26 illustrates this example.

**Extending Aladin's features**

By enabling communication with other applications which expose their supported messages, PLASTIC extends Aladin's capabilities. For instance, Aladin can query a SSAP (Simple Spectrum Access Protocol) server, and display the result list of available specta but can not display any spectrum. Using PLASTIC, Aladin is able to look for a spectrum viewer (eg. SPLAT-VO or VOSpec) and to visualize selected spectra in this application. (see Figure 27).

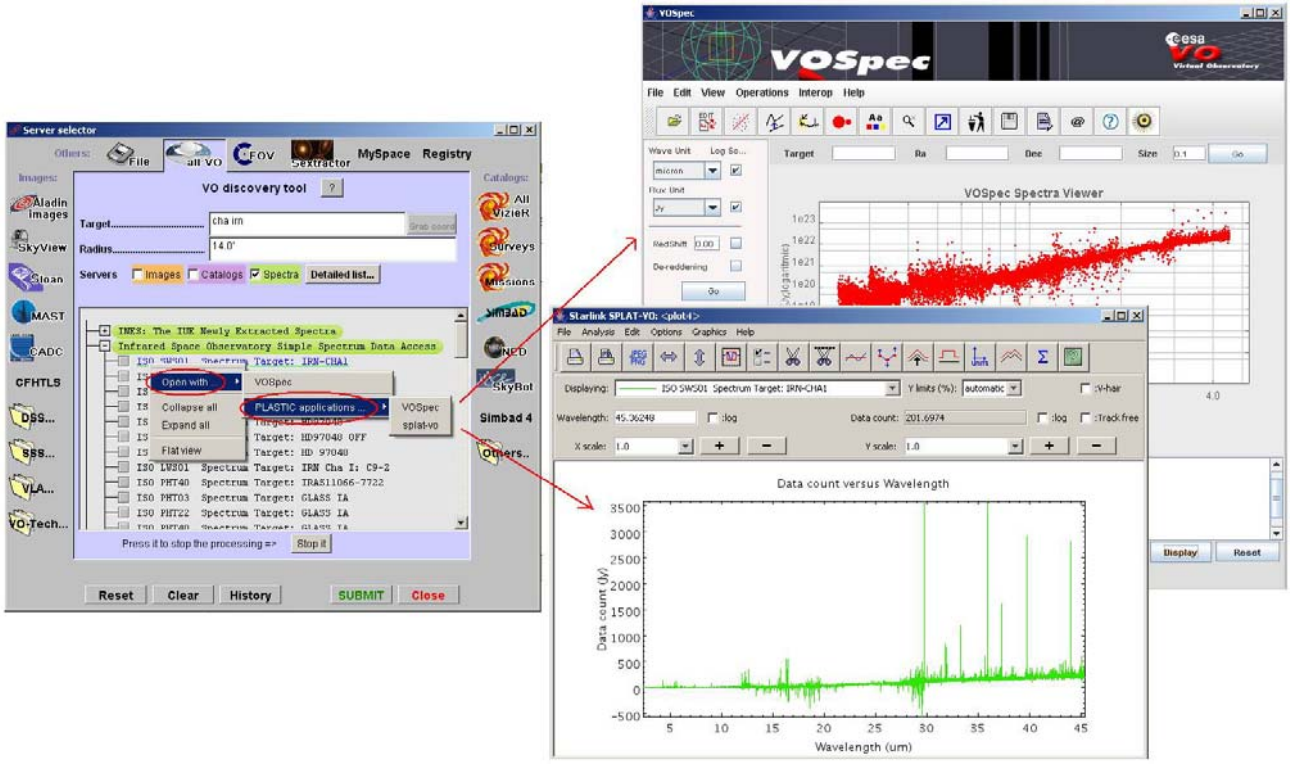


**Figure 27: Sending URLs of spectra from Aladin to VOSpec and SPLAT**

**Using Aladin to extend its own application features**

In a similar way, other applications can take advantage of Aladin's features, notably displaying images and overlaying catalogues. When designing a new application, the developer can consider Aladin as a building block he can use at his ease, with the Aladin API being the list of messages Aladin supports:

- load a VOTable document (from a URL or directly from a string)
- load a FITS image from a URL
- select a list of objects in a table
- highlight a given object
- show a given position on the sky

Using the Aladin plugin mechanism (described in the *Advanced usage* section below), this list of supported messages could even be extended by third-party developers.

This ability to pilot Aladin is available for Java, but also for any language having support for XML-RPC.

Example: The eStar [57] prototype event monitor makes use of Aladin (through PLASTIC) in order to visualize the position on the sky of VOEvent packets (see Figure 28).

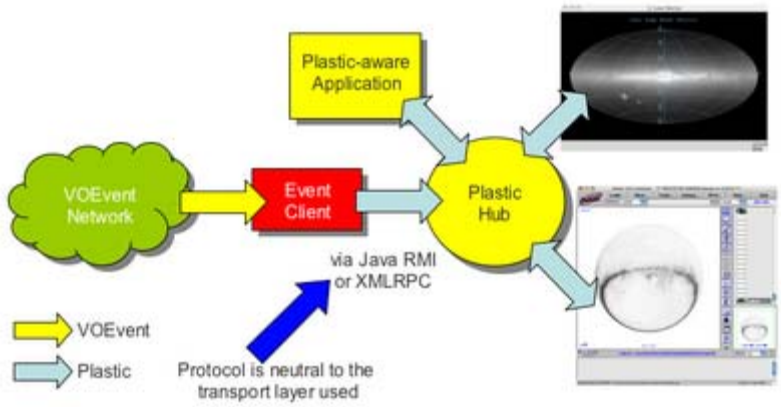


**Figure 28: The eSTAR PLASTIC-based event monitor**

## Cross-selection of data

One of the most promising and powerful feature brought by PLASTIC is the ability to visualize simultaneously selections of objects in different collaborative applications. Those linked views across applications are quite useful to explore a dataset, to see it from different perspectives. For instance, when using together Aladin, VisIVO and TOPCAT, the user is able to select a subset of sources in the parameter space and to see where they lie in the physical space (i.e. on the sky). Conversely, selecting some sources located in a given region of the sky will show in TOPCAT and VisIVO their location in a 2D or a 3D plot.

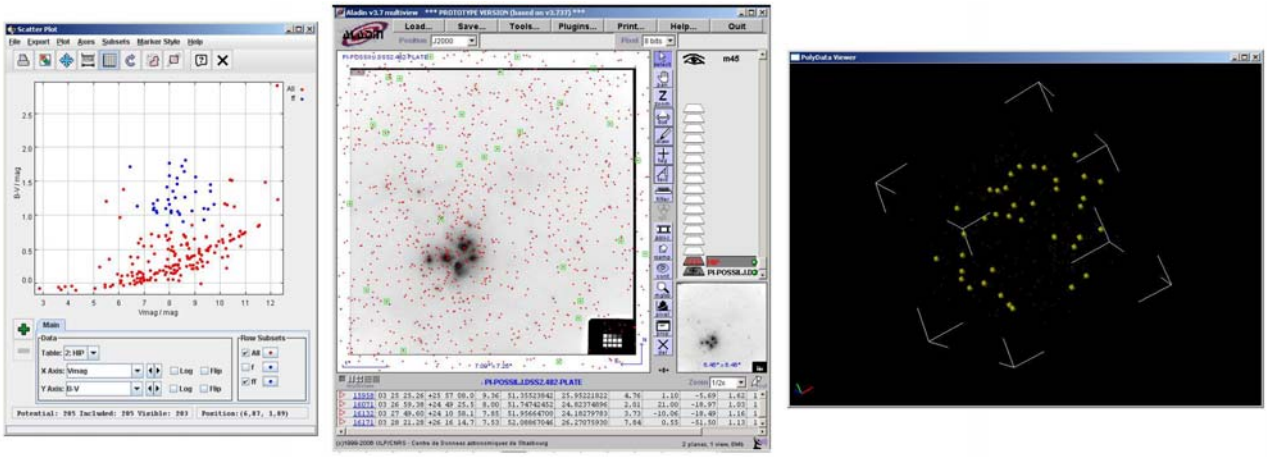

**Figure 29: Linked views between TOPCAT, Aladin and VisIVO**

In Figure 29, a subset of sources has been selected in a 2D graph in TOPCAT. Corresponding sources have been selected in Aladin, and in VisIVO 3D plot. A Wiki page [58] gives an overview of how TOPCAT and Aladin can collaborate via PLASTIC.

## Advanced usage

The Aladin v4 release will include a mechanism allowing users to develop their own plugins. When combined with PLASTIC, this plugin mechanism can lead to powerful developments. Figure

30 shows such an example. When clicking on a pixel of a data cube, the plugin extracts the corresponding spectrum, connects to PLASTIC, and asks VOSpec to visualize the created spectrum.

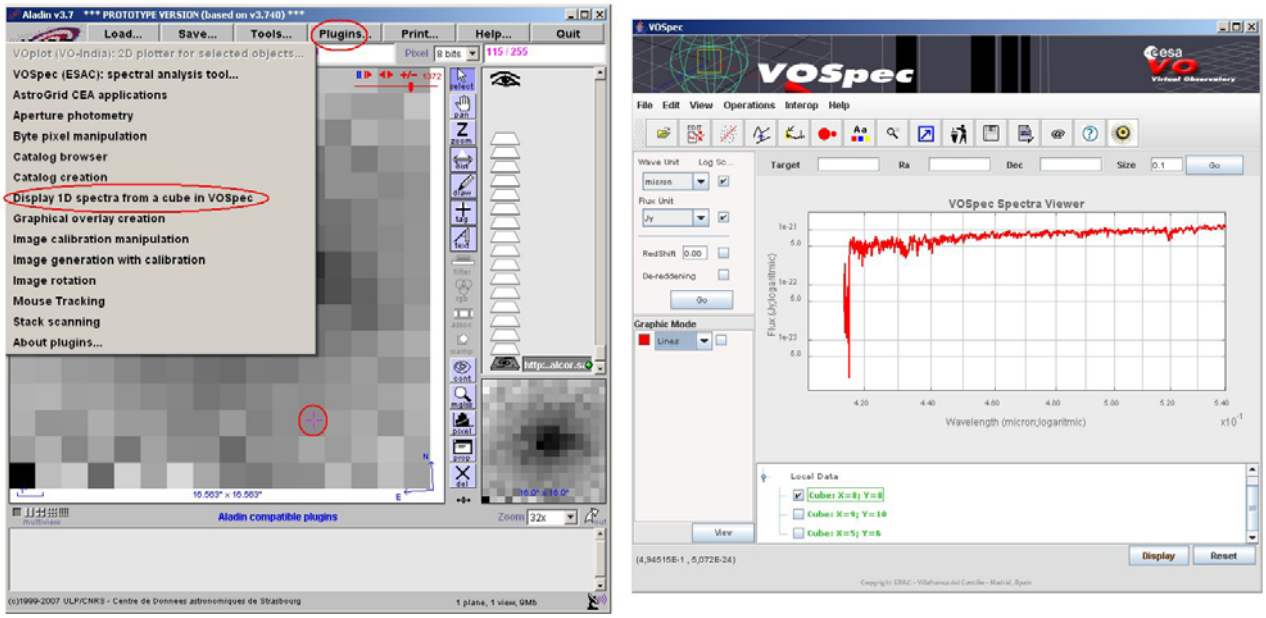


**Figure 30:An example of an Aladin plugin using PLASTIC**

## 5.4.5 Aladin: metadata visualization

In addition to the extensions to Aladin made possible through PLASTIC, and discussed in the previous section, the DS6 team has been involved in enhancing Aladin's metadata visualization capabilities, in the light of IVOA standards being developed by the Data Access Layer (DAL) and Data Modelling (DM) Working Groups.

### 5.4.5.1 Metadatatree and DAL extensions

#### DAL query response visualization

In the VO, available image data services are made known to potential users through registries. Images can then be retrieved and displayed in VO portals such as Aladin (or GAIA, DS9, etc ...) DAL standards (e.g. SIAP) dictate that image services should obey a two step protocol: the actual retrieval of data following a query phase, the response of which describes each dataset matching a given constraint. The basic SIA protocol defines standard query parameters for an image service as well as a single <TABLE> in VOTable format with a couple of standard description <FIELD> elements for the query response.

Any image-oriented VO portal such as Aladin should be able to display these tabular lists of metadata, in order to help users to discover appropriate data. Aladin developed this functionality immediately after the SIA protocol was first described, in the autumn of 2002. (See Figure 31.)

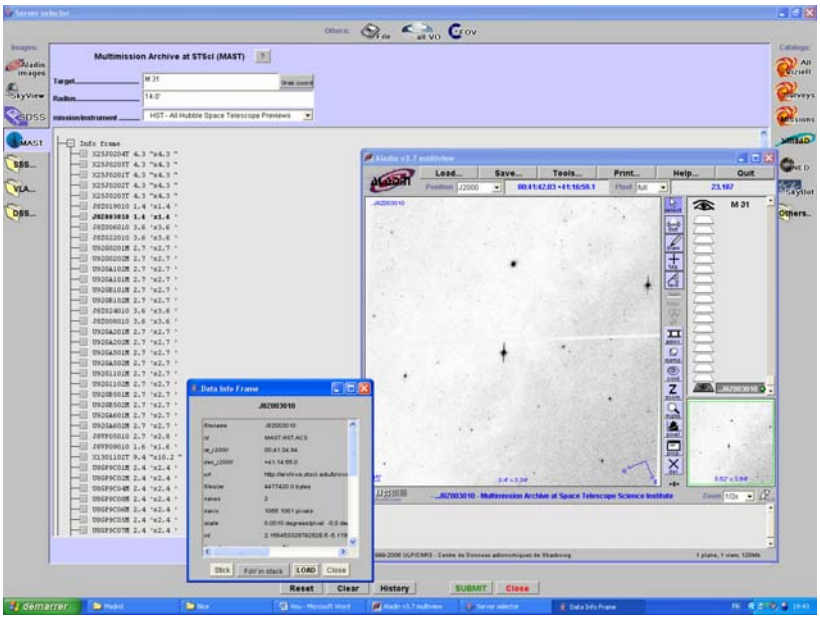

**Figure 31: List of datasets and metadata for an image displayed in Aladin**

## Metadatatree

As early as the first year of the AVO project work was started to think how to organize, and make best use of, metadata for intelligent data discovery in Aladin. This was relying not only on the first SIA attempts but also on a specific CDS VOTable format for query response called IDHA, based on the IDHA data model [59]. Metadata describing the spatial limits, or size, of the datasets are used to display contours of the observation Field of Views on Aladin Previews.

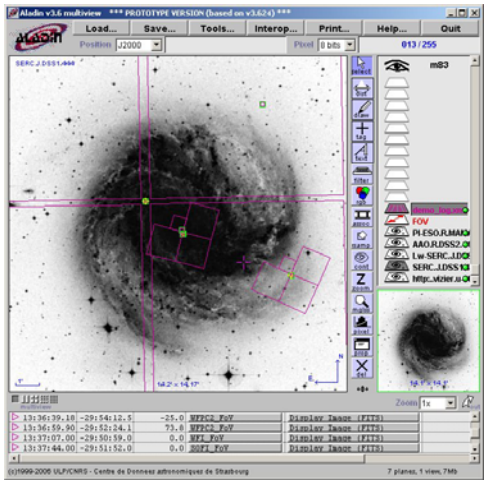

**Figure 32: Fields of Views of observations plotted on top of a preview imagein Aladin**

The idea was to be able to organize the metadata visually in order to help the user to pick up the information he needs more conveniently. From this came the concept of metadatatree where metadata describing datasets are sorted out along the branches of trees defining specific data collections, instruments, filters or colors, epochs, etc. Generic information associated to each node can be displayed as well as metadata specific to each dataset level.

The DAL simple protocols have the drawback of creating different records for each access mode or view of a single dataset, which may make it difficult to clarify differences between, for example, a low resolution preview, a cutout in FITS and a graphical output. The potential number of access modes may become really large when facing more general "imaging" datasets such as cubes or 4D data (in this context a "cutout" can be a single 2D image, a 1D spectrum or a subcube). For all of these reasons it would be nice to have one single node for the description of the dataset and a leaf for every possible access mode under that node

**DAL query extensions**

To achieve this, given the relatively unstructured nature of the metadata in an SIA query response, the CDS team within DS6 has introduced the concept of DAL query extensions. These have been accepted, in principle, by the DAL Working Group, and should be included in the next SSA working draft.

*Use Cases*

There are plenty of Use Cases for using extensions in the DAL query response, in each of which the standard single response is felt to be insufficient. Use Cases include the following circumstances:

- There is a large amount of metadata common to collections or sub-collections
- Complex relationships exist between observations, such as in individual HST observations and so called "HST associations".
- Heterogeneous data relationships, such as images plus associated spectra or source catalogues
- Observations with complex structure, such as *N* observations in multiple bands and exposures.
- Various access and views modes to the same dataset
- Provenance (software and instrumental) and calibration details
- Full IVOA characterisation of a dataset, including sophisticated spatial field of views.

*Mechanism*

A DAL query response VOTable document contains a main Resource the name of which is "results". Extensions will consist of one to several additional resources whose names, ids, types are free. Traditional DAL software clients will only read the "results" resource, but the internal and sometimes complex data structure could be represented and used by new SSA/SIA software clients.

Indexing mechanism:

To create links between records in the main table and additional information contained in the additional resources, it is allowed to add one or more <FIELD> elements to the main table with references to elements situated in the additional sections (free names and ids, utype ='a data model exact name for this field' – where the utype can help a client software to extract the matching information). It is also allowed to refer to elements in the additional section from a standard SSA/SIA field. The *ref* mechanism is not sufficient to create the links between records in the main table and the additional information, because each record in the main table will generally need a specific extension different from those of the other records. That is why we need an indexing mechanism. The values taken by the extension field(s) for specific records are used as keys for this indexing mechanism.

Utypes and software behaviour:

Utypes placed on the Ref/extension Fields are borrowed from the IVOA, or ad hoc, data models. They qualify accurately the field and guide the behaviour of the software towards its content.

## Implementation in Aladin

One of the first extension use cases Aladin implemented was the CGPS archive navigator, developed in collaboration with Daniel Durand at CADC. Figures 33 to 37 show the metadatatree with access to the full data retrieval page, an image preview, with access to one extracted single spectrum, and access to one lambda plane. On each of these screenshots the access mode leaf in the tree is in bold face.

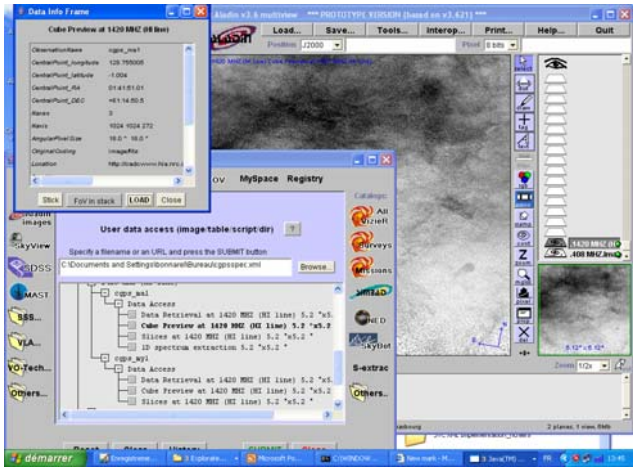

**Figure 33: The datatree for a cube, with the cube preview displayed**

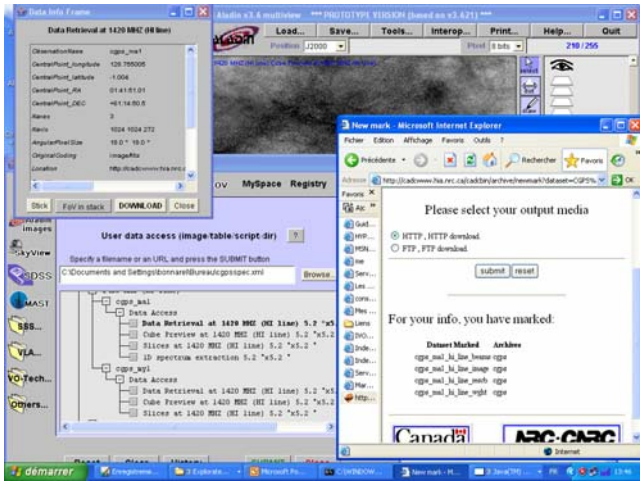

**Figure 34: The data retrieval page**

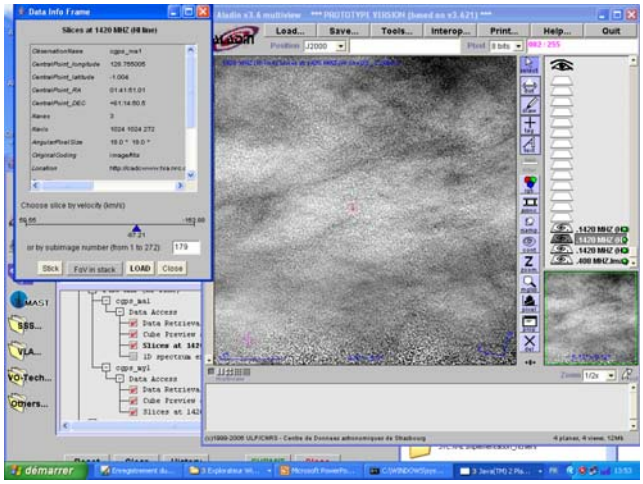

**Figure 35: Lambda plane access**

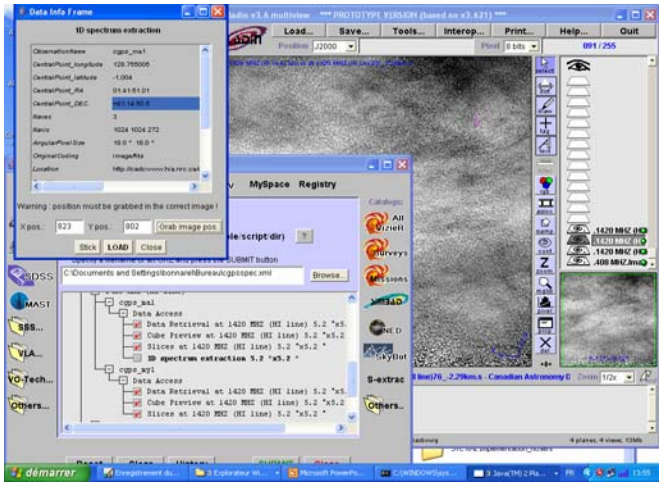

**Figure 36: Selection of spectrum position**

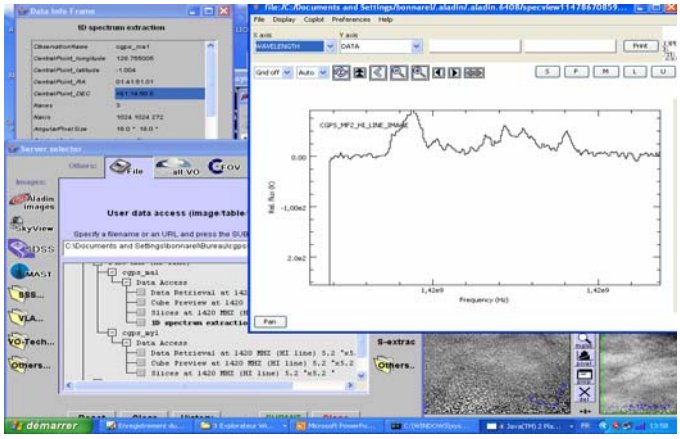

**Figure 37: Spectrum visualization**

## 5.4.5.2 Dataset Characterisation

The first generation DAL protocols like SIA1.0 allow some raw description of datasets including curation and provenance information, as well as more data-oriented characterisation, such as bandwidth, central position of observation in the sky, observation date and time, etc. But this was incomplete and actually not well structured, so one of the initial goals of DAL working group was to rely on standard data models to describe observations and datasets.

The IVOA developed an *Observation* data model dedicated to spectral data - the so called "IVOA Spectrum data model". For more general datasets the *Characterisation* data model describes the dataset in the data parameter space and the Spectrum data model characterization class is consistent with the generic IVOA characterization data model. XML and VOTable implementations of the data models have been defined in IVOA Working Drafts for these two models and attributes of the model are universally designated by character strings called “utypes”.

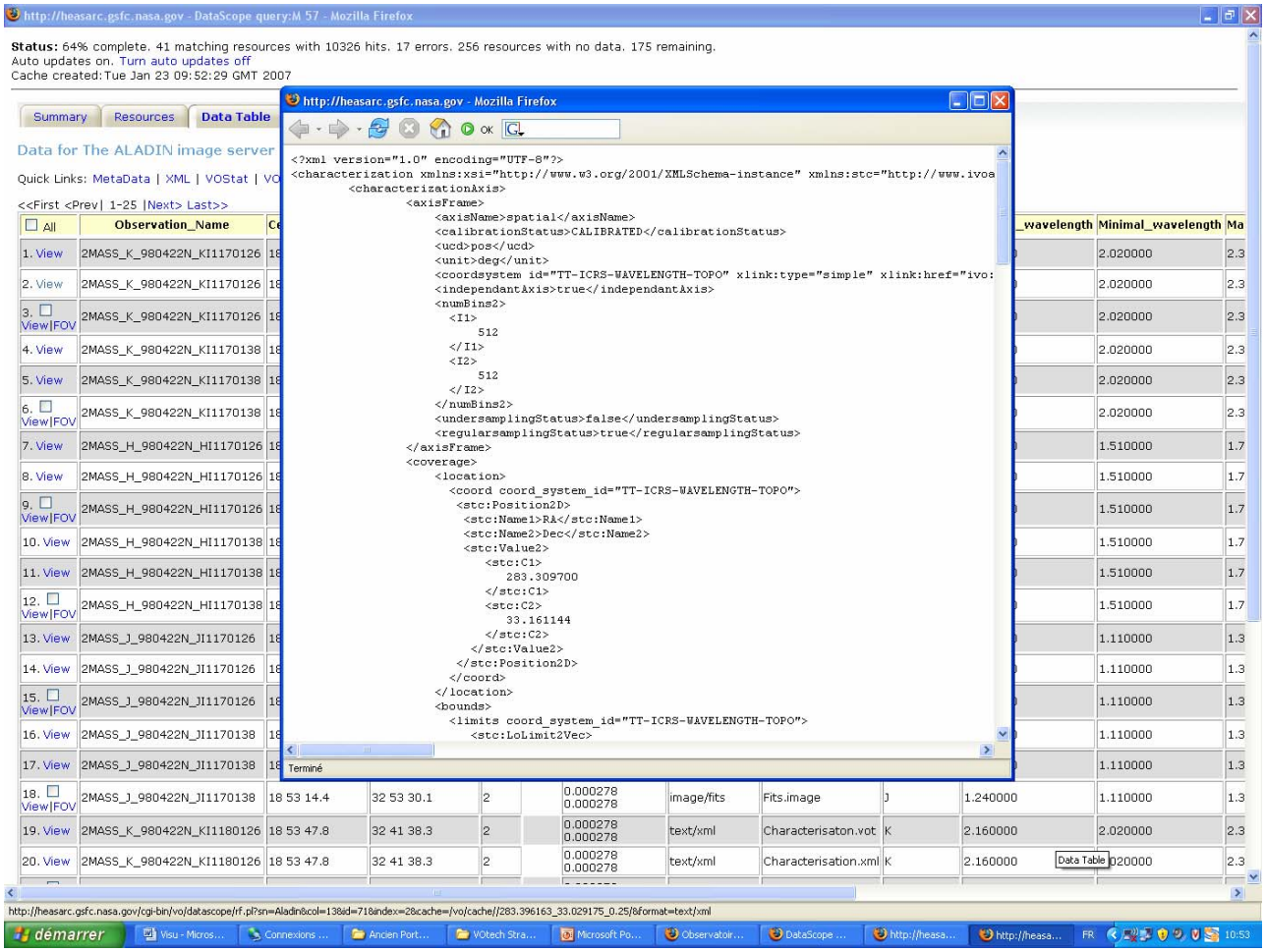


**Figure 38: Characterisation of a dataset: output from the Aladin image server in Datascope**

SSA 1.0 reuses the spectrum and characterization data model by standardizing metadata in the query response, using utypes to define the response <FIELD> elements.

The Characterisation data model organizes the dataset description along three "directions":

- The *property* distinguishes the data coverage, from the data resolution and the data sampling.
- The *axis* direction distinguishes specific quantities in the data such as spatial, time, spectral ones, etc.
- The *level* direction describes the degree of granularity with which the other two are given from level 1 for a raw location of the data to level 4 giving an accurate mapping of the data.

The DS6 team has participated to the development of the IVOA Characterisation data model, and a first implementation in the Aladin image server has been released and interfaced with the SIA interface of the Aladin server, as illustrated in Figure 38.

# 5.5 Addressing scalability issues

As discussed above, scalability is one of the major issues for data exploration in the VO, and in this Section we discuss some of the work undertaken addressing scalability issues to date in DS6.

## 5.5.1 Scalable access to tabular data

### 5.5.1.1. Introduction

Data exploration within the Virtual Observatory is in large part the investigation of tabular data, usually source catalogues of some kind. Some of the data sets we wish to investigate, especially the most interesting ones (surveys with wide coverage and deep observations) are already very large; for instance the main photometric catalogue of the SDSS DR5 contains ~$10^8$ rows and 446 columns - around 0.5Tbyte.

As astronomical and computing hardware becomes more sophisticated these sizes will only increase. The processor speeds and memory and disk capacities and bandwidths available on the machines astronomers use for exploring these datasets are increasing too, but are likely always to lag behind. Investigating these data therefore presents a major challenge for astronomers and the software which serves them.

The quantitative issues of processing time for various operations on such datasets has qualitative implications for the science which can be done with them. For thoroughly-understood data, it may be that only a single colour-magnitude plot is required, and whether this takes a second or a day to produce is of minor significance, since the time to generate the catalogue in the first place will far exceed either. However, for data which are less well understood, for instance because it contains objects or

quantities invisible to previous observations, a far more interactive style of analysis is required, especially if unexpected relationships are to be discovered. For these cases, a tool which can perform plots, column calculations and row selections `interactively' (in a matter of a few mouse clicks and a few seconds) will allow the perceptive astronomer to uncover features and relationships in the data in a way which would simply not be practical using software that works in `batch mode'. One characteristic data access pattern, which tends to be the bottleneck, in this kind of interactive investigation is repeated scans of a few columns in their entirety. For the purpose of this report, we consider datasets which are static and read-only.

This section discusses approaches to these challenges and describes software which has addressed them, presenting the results of some relevant benchmarks. The scalability features are implemented in STIL [60], which is a general purpose library for table I/O. The main application packages currently built on this are TOPCAT, which is a powerful GUI application for exploration and analysis of (particularly astronomical) tables, and STILTS, which is its command-line counterpart.

### 5.5.1.2 Current Practice

As regards access to persistent tabular (or other) data there are two basic approaches to writing software:

- Load the data into memory and then process it
- Keep the data on disk and perform random access directly

The first approach is to ingest an entire dataset into memory from some standard or custom format on disk (or from the network or as the result of other processing), and then perform processing on it. This is generally easiest to write and fastest to run for small data sets, and most existing tabular processing tools, environments and visualisation packages fall into this category. It is not well suited however to datasets which, when loaded, are larger than the available memory of the machine running the software. Languages which can make use of virtual memory[4] may be able to process datasets larger than the machine's physical memory, but this can often result in poor performance. Since, at time of writing, it is not uncommon to have 1 Gbyte or more of physical memory in desktop or laptop systems, this can in principle allow access to reasonable sized datasets - up to a few hundred million table cells *if* the application is written with this kind of scalability in mind, although often it is not. Note however that the start-up time for an application which loads hundreds of megabytes from disk to memory will be rather slow, quite possibly several minutes.

The second approach has largely been the preserve of Relational Database Management Systems (RDBMS). These are powerful general purpose software systems which can take datasets of astronomical magnitude in their stride, and they are in fact used in practically all cases for data management of the large astronomical surveys. For the purposes of data exploration these systems have some important

[4] This does not in practice include current implementations of Java. Although available heap memory can be configured at run time, setting this to a value greater than available physical memory invariably results in disk thrashing and dismal performance. The reasons for this are presumably related to Java's garbage collection algorithms.

drawbacks. In the first place, data access is exclusively using some variant of the Structured Query Language (SQL), which makes it difficult to extend their operation, for instance by adding interactive graphical capabilities, rich metadata handling, or astronomically specific features such as fuzzy crossmatching. In the second place, while they make many operations extremely fast, others can be disappointingly slow. RDBMS rely for their efficiency partly on precomputed indices attached to table columns, but there are several important operations for which such indices cannot reasonably be precomputed (since it is not obvious which ones it would be necessary to supply) or cannot be used even if they are available (since a scan of every value in the column is necessary). Examples of some of these unoptimised/unoptimisable queries are given in Section 5.5.1.6.

Some other astronomical software takes this second approach as well, though in many cases it is custom code written by individuals to perform fairly specific tasks on fairly specific data.

In practice therefore, to explore data from a dataset too large to load into memory, a hybrid approach is usually used. An astronomer first performs some batch-like step to assemble a reduced dataset of modest size from the full dataset, which may be stored locally or at a remote data centre. The reduced set can then be investigated using the facilities of some interactive tool or tools. The initial reduction can be specified in various ways: in the most straightforward case rows can be preselected using some criterion such as sky position, magnitude or colour, or some combination of such criteria, and/or columns which are not required can be discarded. Other techniques such as clustering or random sampling may also be used.

In many cases, for instance spatially limited studies where a natural data subset of manageable size exists, this hybrid approach is quite satisfactory. In others however the requirement to cut the initial dataset down to size can result in discarding data of interest. This is particularly the case for more speculative exploration of a catalogue, where sources of unexpected types, or the information required to identify them as such, may inadvertently be thrown out at an early stage. Moreover the preselection step itself may be slow, which will discourage experimentation with different reduced datasets.

### 5.5.1.3 Aims

In view of the above problem, this report discusses how the following related challenges can be addressed:

- To improve data access times for large datasets
- To maximise the size of dataset which can be investigated interactively

Interactive exploration software such as TOPCAT is characterised by a mixture of full-column scans and scattered random access, so we are seeking a way of providing both access patterns as efficiently as possible.

The solutions discussed here have been implemented, and are available for scientific use, in the STIL library and TOPCAT and STILTS applications.

### 5.5.1.4 Techniques

This section discusses two of the data access techniques used by the software reported here. Neither of these is particularly novel in astronomical contexts; column-oriented table handling in the MIDAS[61] and STSDAS[62] packages and file mapping in the Starlink HDS[63] library provide examples of prior art of a couple of decades' standing. However, the importance of these techniques in supporting the reported performance at high data volumes makes it worthwhile to explain them in some detail.

**Column-oriented Access**

There are two obvious ways of storing the data of a table in a sequential form (such as a single disk file). The first is `row-oriented', in which all the cells of the first row are stored first, followed by all the cells of the second row, and so on. The second is `column-oriented', in which all the cells of the first column are stored first, followed by all the cells of the second column and so on.

For many purposes, it does not matter which of these schemes is used. A program which has to read every cell of the table from disk at once (for instance to ingest it into memory prior to providing graphical facilities, or to copy or transmit it entire to some other subsystem) is likely to work just as well in either case. However, a program which needs only to access a few rows may have better performance with row-oriented storage, and one which needs only to access a few columns may have better performance with column-oriented. The reason is twofold: firstly, reading bytes from disk can be done much faster if they are contiguous than if they are scattered in different places. Secondly, contiguous bytes will span many fewer disk blocks/memory pages than scattered ones, which means that the OS stands a better chance of keeping them in cache to obviate the need for subsequent disk accesses if the same data are required again.

Most existing general purpose tabular storage formats are row-oriented. This is the case for FITS binary and ASCII tables, all variants of the VOTable format, and most ASCII-based formats such as comma-separated values (CSV). Much of the software which uses these formats reads the whole table prior to processing, so there is no great performance advantage either way.

It turns out that many of the most expensive data access patterns required for the kind of interactive processing we want to do on large datasets need to access all the cells in just a few columns of a table. These include graphical operations such as generating scatter plots, histograms or density maps, statistical operations such as calculating univariate or bivariate measures such as means and regressions, and preparing short-lived indices for subsequent operations on row selections. In these cases data access can be performed much more efficiently if the disk file is arranged in column-oriented fashion.

It is worth discussing the position of RDBMS in this context. Nearly all RDBMS use row-oriented storage. This may be partly because it makes row insertion and deletion easier to do, but since we are regarding our datasets as static for processing purposes these are not operations we require. At least one column-oriented RDBMS does exist, Sybase-IQ[64], but it is a specialised product with very high licensing costs which makes it unsuitable for most astronomical applications. To achieve the high performance which they exhibit for row selections and other query operations,

RDBMS make use of pre-computed persistent indices attached to columns. To locate a selection of rows which obey defined bounds on pre-indexed columns, this works very well, so if column *J* is indexed, then

```
SELECT * FROM PSC WHERE J > 18.5
```

will execute very quickly even on a large table, assuming the number of rows in the selected set is small.

However, selecting on an unindexed column, or combination of columns, will require a scan of all the values, so

```
SELECT * FROM PSC WHERE J - K > 5.0
```

will be slow even if *J* and *K* are indexed. In principle it is possible to index (*J* – *K*), but clearly one cannot cater for all possible combinations of columns. Other operations on a full dataset, such as calculating the mean or variance of a column, generating a plot, or computing a new index, inherently require a scan of all the cells in a column, indexed or not. In these cases, where pre-computed indices are not present or cannot be used, RDBMS fall foul of the problem of column-oriented access to row-oriented data and exhibit the expected poor performance. These problems become more acute the wider a table is, *i.e.* the more columns it has. By using column-oriented access therefore, it should be possible to outperform the RDBMS significantly on such operations.

**File Mapping**

As mentioned in Section 5.5.1.2, the alternative to reading an entire dataset into memory from disk prior to processing is to read items (such as table cells or rows) from a random-access disk file as they are required. A naïve implementation of this kind of access, using a seek followed by a read to obtain each item, typically results in very slow I/O, since seeks and small reads have high overheads. Operating systems which perform automatic caching of data read from disk can ameliorate, but not eliminate, this problem. To perform disk-based random access efficiently, it is necessary to ensure that actual reads are, as far as possible, large and infrequent. For simple and predictable access patterns it is fairly easy to arrange this, for instance a sequential read of all the rows in a table from start to finish can be done using buffered reads of large blocks of bytes from disk at a time. Where processing algorithms require unpredictable patterns of random access much more work is required to achieve efficient direct disk I/O however, involving caching of buffered blocks in memory with associated cache block replacement strategies and so on. Some applications, probably including most RDBMS, implement such strategies which are highly tuned to the data, and possibly the operating system, for which they are designed, but a high quality implementation of this kind of caching system is a research project in itself.

Fortunately, most operating systems provide a mode of disk access known as *file mapping*, which provides these features in a way which is easy for user processes to utilise. When a disk file is mapped, its bytes appear in a process's address space as if

they had been stored there by memory write commands. The content of the file therefore looks much like the content of a byte array in memory. This can be achieved using the `mmap(2)` call in Unix, or the `java.nio.channels.FileChannel.map()` call in Java (J2SE1.4 and later).

There are a number of features of this technique which make it suitable for accessing large datasets of the kind we are interested in here:

- Establishing the mapping is effectively instantaneous, even for large files, unlike reading a large file from disk.
- Data access is typically somewhat faster than for normal reads from disk.
- If only a small amount of the data (a few table cells) is required, only that much is read.
- Small reads do not incur unavoidable overheads.
- The total size of the file (or rather of its mapped region) is not limited by the size of physical memory or of swap space, though it is limited by the size of unused addressable memory (see below).
- The operating system is able to cache disk blocks in physical memory using its own algorithms.

The final point, as discussed above, is somewhat dependent on the details of the OS implementation, but in many cases it can provide great advantages. It generally means that both sequential and random access to the bytes of the file are performed in a highly optimised fashion, in a way which would be hard work for the programmer to implement at application level. One practical effect is that if the mapped file is smaller than available physical memory, each block is only read from disk once (when it is first accessed) during the lifetime of the application, and subsequent accesses are supplied from memory cache at very low cost. This effectively gives performance characteristics similar to the read-then-process approach but without the slow start-up. Even when this is not the case, many access patterns will end up much faster than they can easily be coded by hand. The application programmer does not need to worry about which of these regimes the program will run in; in either case the OS can generally be relied upon to arrange the I/O in an efficient fashion.

The facility for memory mapping is not available in all languages or all operating systems, but it can be done in C and related languages and in Java on most important OSs. It is particularly suitable for use with Java, since, at least in current implementations, Java otherwise has poor support for using large amounts of virtual memory.

A couple of drawbacks of this technique should be mentioned. One is that to use any kind of random access on sequential data, mapped or otherwise, it must be possible, and preferably fast, to calculate the offset of each item within it. This is easy to arrange for tabular data in which each cell in a given column has a fixed size in bytes, but it does present difficulties for tables with variable-length cells or rows, and it precludes the use of most kinds of disk file compression, as well as free-form text formats including XML. Secondly, the total size of files mapped by an application, or at least of the mapped regions of such files, is limited by the unused address space available to a process. For a 64-bit operating system this is millions of Terabytes and is unlikely to present a problem for the forseeable future, but for 32-bit OSs this figure will be <4Gbyte, which can represent a serious limitation for large datasets. The more

ambitious operations reported in Section 5.5.1.6 could therefore only be run under a 64-bit OS.

### 5.5.1.5 Implementation

We have experimented with the ideas discussed in the previous Section to write some table I/O handlers for STIL, the Starlink Tables Infrastructure Library. STIL provides a useful laboratory (as well as production system) for this kind of work, since it allows use of pluggable data handling backends. Applications built on top of it (in particular the GUI application TOPCAT and the command-line table processing package STILTS) can be run using different table I/O handlers. STILTS provides a set of facilities broadly comparable with those available using SQL, so that similar operations can be performed using both STILTS (with various different handlers) and an RDBMS, and the performances compared. Unlike the RDBMS, this software provides no *persistent* indexing facilities. Index-like structures are used in various processing algorithms, but in each case the index is computed as part of the processing step and discarded at some point before the application exits. This reflects the focus of STIL on permitting free-form exploration of the data as opposed to the focus of RDBMS on preparing it for certain common types of query.

Two file-mapping based handlers have been written, one row-oriented and one column-oriented. Both of these were based on the FITS BINTABLE format. Some efficiency might have been gained by inventing custom binary formats, but FITS fitted the requirements quite well, and the compatibility with existing astronomical software is a bonus.

The row-oriented variant is a perfectly normal FITS BINTABLE with `NAXIS2` equal to the number of table rows. This format is designated ``fits" below.The column-oriented variant is a BINTABLE consisting of a single row (`NAXIS2`=1), with each cell in that row being an array which contains a whole column's worth of data. This is a perfectly legal FITS file, but it may not be well understood by other `normal' FITS-aware software. This format is designated ``colfits" below. In each case column names, units and types are stored using `T***n` header cards in the usual way. These formats satisfy the requirements for cell data organisation on disk of row- and column-oriented formats respectively. The I/O software was largely custom-written, and there was no problem in either the file mapping (provided a 64-bit OS was used) or FITS header processing with files containing tens of Gbyte and hundreds of millions of rows (though general purpose FITS software might have trouble).

A disadvantage of FITS over VOTable format is that it cannot hold such rich metadata (this is one reason why VOTable was invented in the first place). To take care of this the I/O handlers permit an optional VOTable describing the FITS table to be stored as a byte array in the primary HDU of the FITS files, while the BINTABLE is stored in the first extension HDU. Any software reading the FITS files can safely ignore the VOTable in the primary HDU if it wishes, since the essential table metadata is stored redundantly in the header of the first extension (BINTABLE) HDU. The handler variants which store additional metadata in this way are called

``fits-plus" and ``colfits-plus" in the STIL package. These arrangements for metadata have no effect on the efficiency concerns discussed in the rest of this report.

### 5.5.1.6 Results

For comparison purposes similar operations were performed in the following ways:

- MySQL: MySQL 4.1.20[65] using an unindexed MySQL MyISAM table
- colfits: STILTS with mapped column-oriented FITS-based storage
- fits: STILTS with mapped row-oriented FITS-based storage
- fits-nomap: STILTS with unmapped row-oriented FITS-based storage

The MySQL table was unindexed because the benchmarks performed here would not have benefited from any indexing. The Java Runtime Environment used for running STILTS was version 1.5.0_10 of Sun's 64-bit J2SE. In all cases a quiescent 3GHz Xeon running 64-bit Linux RHEL4 (kernel version 2.6.9) with 2Gbyte of memory and a single Serial ATA disk was used.

There were some unexpected issues with Linux virtual memory management when STILTS was mapping file regions several times larger than physical memory; without additional tuning, this led to swap thrashing which increased benchmark times considerably and led to serious degradation of other concurrent processes.
Tuning kernel parameters improved matters somewhat[5] but only switching off swapping altogether (`swapoff -a`) fixed the problem completely. Since this rather drastic measure may be undesirable in some operating environments it is hoped that future versions of the Linux kernel will provide improved performance under these conditions, or at least better VM tuning options. Other operating systems may perform better. Or worse. Note that these memory management issues only arose for very large datasets, more likely to be found at data centres than on desktops - we do not expect the average astronomer to have to consider such esoteric matters for normal use.

The following test tables were used:

- **XSC**: 2MASS[66] Extended Source Catalogue (1,647,599 rows x 391 cols, ~2.2Gbyte)
- **XSC_B**: Northern-hemisphere subset of XSC (908,817 rows x 391 cols, ~1.2Gbyte)
- **PSC**: 2MASS Point Source Catalogue (470,992,970 rows x 61 cols, ~111Gbyte)
- **PSC_B**: Northern-hemisphere subset of PSC (177,756,896 rows x 61 cols, ~42Gbyte)

[5] Some values in the `/proc/sys/vm` directory were changed: setting `swappiness` to 0 helped though didn't eliminate swapping altogether; tweaking `percpu_pagelist_fraction` and `min_free_kbytes` may or may not have had an effect.

These probe four different large-dataset regimes in relation to the 2Gbyte of RAM on the benchmark machine: for XSC_B the whole file fits in physical memory, for XSC nearly the whole file fits in memory, for PSC_B a couple of columns fit in memory, and for PSC not even a single column does.

The following benchmark queries were run.

- **X(B)_SEL2**: Select for J_M_FE - K_M_FE > 8 in XSC(B)
- **X(B)_STAT1**: Statistics on J_M_FE in XSC(B)
- **X(B)_STAT2**: Statistics on H_M_FE and K_M_FE in XSC(B)
- **P(B)_SEL2**: Select for J_M - K_M > 8 in PSC(B)
- **P(B)_STAT1**: Statistics on J_M in PSC(B)
- **P(B)_STAT2**: Statistics on H_M and K_M in PSC(B)

All these cases represent astronomically reasonable queries on the data, and do very much the same thing in the two software systems, in a way which is as efficient as can reasonably be done.

The actual commands used for MySQL and STILTS respectively are given in the following two tables.

| **Bench** | **STILTS command** |
|---|---|
| X_SEL2 | `stilts tpipe cmd='select j_m_fe-k_m_fe>8.0' cmd='keepcols "designation h_m_fe j_m_fe k_m_fe"'` |
| X_STAT1 | `stilts tpipe cmd='keepcols j_m_fe' omode=stats` |
| X_STAT2 | `stilts tpipe cmd='keepcols "h_m_fe k_m_fe"' omode=stats` |
| P_SEL2 | `stilts tpipe cmd='select j_m-k_m>8.0' cmd='keepcols "designation ra decl h_m j_m k_m"'` |
| P_STAT1 | `stilts tpipe cmd='keepcols j_m' omode=stats` |
| P_STAT2 | `stilts tpipe cmd='keepcols "h_m k_m"' omode=stats` |

**Table 2: STILTS command for executing benchmarks**

| **Bench** | **MySQL command** |
|---|---|
| X_SEL2 | `SELECT designation, h_m_fe, j_m_fe, k_m_fe FROM xsc WHERE j_m_fe - k_m_fe > 8.0` |
| X_STAT1 | `SELECT COUNT(j_m_fe), AVG(j_m_fe), MIN(j_m_fe), MAX(j_m_fe) FROM xsc` |
| X_STAT2 | `SELECT COUNT(j_m_fe), AVG(h_m_fe), MIN(h_m_fe), MAX(h_m_fe), AVG(k_m_fe), MIN(k_m_fe), MAX(k_m_fe) FROM xsc` |
| P_SEL2 | `SELECT designation, ra, decl, h_m, j_m, k_m FROM psc WHERE j_m - k_m > 8.0` |
| P_STAT1 | `SELECT COUNT(j_m), AVG(j_m), MIN(j_m), MAX(j_m) FROM psc` |
| P_STAT2 | `SELECT COUNT(j_m), AVG(h_m), MIN(h_m), MAX(h_m), AVG(k_m), MIN(k_m), MAX(k_m) FROM psc` |

**Table 3: MySQL commands for executing benchmarks**

| Benchmark | MySQL | | colfits | | fits | | fits-nomap | |
|---|---|---|---|---|---|---|---|---|
| X_SEL2 | 66 | 49 | 4.7 | 3.2 | 89 | 86 | 449 | 442 |
| X_STAT1 | 65 | 49 | 2.0 | 1.8 | 51 | 39 | 208 | 223 |
| X_STAT2 | 66 | 49 | 2.4 | 2.1 | 51 | 44 | 224 | 221 |
| XB_SEL2 | 36 | 27 | 2.4 | 2.1 | 21 | 3.8 | 124 | 124 |
| XB_STAT1 | 36 | 27 | 1.4 | 1.0 | 14 | 2.6 | 117 | 113 |
| XB_STAT2 | 36 | 27 | 2.0 | 1.2 | 14 | 2.8 | 122 | 119 |
| P_SEL2 | 3422 | | 397 | | 2417 | | 10281 | |
| P_STAT1 | 3390 | | 105 | | 2321 | | 10256 | |
| P_STAT2 | 3404 | | 284 | | 2351 | | 10399 | |
| PB_SEL2 | 1278 | | 95 | 74 | 837 | | 3840 | |
| PB_STAT1 | 1330 | | 37 | 28 | 811 | | 3926 | |
| PB_STAT2 | 1290 | | 70 | 59 | 802 | | 3926 | |

**Table 4: Timing results for the benchmarks. Figures given are elapsed times in seconds. Each test was run twice in immediate succession; where the second run was, or might be expected to be, faster than the first, both times are shown. This is expected if enough physical memory is available that the data read from disk in the first run can remain in cache for the second one.**

We see that the colfits results are 10-50 times faster than the MySQL ones for the queries shown here. The shorter (X*) colfits results probably include per-task overheads which could be reduced with a bit of additional coding effort.

The X* and PB_* results, but not the P_* ones, represent a regime in which a few columns fit entirely into physical memory, so that a second similar query can use data cached by the OS rather than requiring further I/O. This does lead to a speedup, though for the column-oriented storage a fairly modest one, showing that these queries are not strongly I/O-bound.

The timings for row-oriented mapped FITS files are also quite good, and better than MySQL, which does not use file mapped I/O. Where the table is small enough to be cached in memory (XB_*), second-run times gain almost an order of magnitude, becoming fast enough for interactive use. Although not shown in the results, running *any* query following an initial one tends to run in the faster time, since the initial scan results in caching the entire table. Row-oriented files larger than physical memory (X_*) however take more than a minute to scan a 2Gbyte file, and will not benefit from caching on subsequent scans.

Timings for row-oriented FITS files using non-mapped access are also shown for reference. These are several times slower than MySQL; less effort has been expended on optimising these. In current implementations of STIL, this unmapped access is the only option for accessing row-oriented FITS files larger than 2Gbyte on a 32-bit platform. Files of this size are a rather specialised interest however, and likely to be accessed only on server-class machines which are more likely to have a 64-bit OS.

The two catalogues used here - the 2MASS XSC and PSC - represent two useful regimes of dataset size. The XSC, at a couple of Gbyte, could reasonably be transferred over the network to users who want to perform detailed analysis on it. In this regime, use of STILTS/colfits, (or STILTS/fits if the file is smaller than physical memory) rather than MySQL permits queries to run in one or two seconds rather than one or two minutes. This brings datasets of this size easily within the range of

interactive analysis, especially given that queries, plots, sorts, crossmatches and so on can be done within the GUI program TOPCAT. Interactive experimentation with TOPCAT confirms this conclusion. Similar comments apply, *a fortiori*, to datasets smaller than the 2MASS XSC.

The PSC, at a hundred Gbyte or so, is really too large for casual acquisition. Moreover, it is not really suitable for use in an interactive tool - elapsed times of a few minutes for typical operations are somewhat longer than desirable for exploratory work, and TOPCAT's current implementation will not cope well with $O(10^8)$ rows. However, the reported benchmarks show that STILTS (which has better streaming capabilities than TOPCAT) can reasonably be applied to some of the largest source catalogues available, and provides performance which is improved considerably over that of conventional RDBMS for certain types of query.

Apart from the virtual memory tuning issues mentioned above, no problem is forseen in applying the same software to datasets several times larger, which covers the largest source catalogues currently available. Limitations of available disk space and data prevented such tests being carried out at this time.

### 5.5.1.7 Conclusions

For large datasets, reading all the data into memory before processing can be unwieldy and often simply will not work. Direct disk access with appropriate use of file mapping provides an efficient alternative without excessive programmer effort, under a wide variety of sequential and random data access patterns. We note however that when mapping extremely large datasets Linux's default virtual memory configuration left something to be desired.

Since many of the most expensive operations which are performed as part of exploratory analysis are effectively scans of the whole of one or a few columns of a table, an analysis application's disk access can be made a great deal more efficient by arranging data on disk in column-oriented order rather than the more common row-oriented order. This is especially the case if the table as a whole is too large to be cached in available physical memory, but a few of its columns together are not; many of the wide tables which characterise modern survey data fit this profile.

Combining file mapping and column-oriented storage can therefore provide very efficient random and sequential access to large tables such as source catalogues, and in particular for certain query types common in exploratory analysis can provide big improvements over the facilities provided by the RDBMS which are traditionally used for access to this kind of data. In the case of tables up to around $10^6$ rows x $10^3$ columns ($\sim 10^0$ Gbyte) queries can be sufficiently responsive to support interactive data exploration -TOPCAT is a GUI program which provides these facilities for such datasets. For much larger datasets (we have benchmarked $10^9$ rows x $10^2$ columns $\sim 10^2$Gbyte) GUI interactive use is so far precluded, but the use of column-oriented storage leads to performance at least an order of magnitude better than common RDBMS for the kind of query in which precomputed indices cannot be used.

### 5.5.1.8 Future Work

We suggest two possibilities for future work prompted by these results, one suited to very large archival datasets and one to medium-sized ones.

*Hybrid Data Servers*

There is of course no suggestion that the software described here should replace the RDBMS for storage and management of very large catalogues, since where precomputed indices can be used, the RDBMS will massively outperform any non-indexed algorithms. Moreover STIL is not well suited to queries on multiple tables linked by keys.

However, a hybrid query engine which can service queries using whichever of the two methods is most appropriate for each case could provide the best of both worlds. This is probably only really worthwhile for rather large (tens of Gbyte) datasets. The data would have to be stored redundantly in both RDBMS and column-oriented format on a server - the low cost of disk storage means this is not a serious problem. Given this setup, one possibility would be to use the RDBMS backend to provide traditional SQL/ADQL-type queries as at present, while using the STIL backend to provide custom column-friendly queries such as calculating full-column statistics, reporting outliers or generating plots and density maps, perhaps driven by a separate HTML form or web-service interface. Another possibility would be to add an SQL/ADQL-like front end to STILTS which could analyse an SQL query to decide whether it could service it efficiently or should pass it on to the RDBMS.

*Medium-Sized Survey Archive*

As seen in Section 5.5.1.6, a survey of the size of the 2MASS extended source catalogue (2Gbyte, a couple of million rows) stored in column-oriented format is quite amenable to interactive investigation using TOPCAT or STILTS, and this is the case even on fairly modestly specified machines. Furthermore, one or two Gbyte is a feasible amount of data to transfer across the internet. This raises the possibility of preparing an archive of column-oriented (colfits) files each containing the complete data of a number of surveys of general interest suitable for the purpose, that is, ones between a few tens of thousands and a few millions of rows in size.

Although datasets in this range are of course already available in various forms, it has before now generally been necessary to perform some kind of pre-selection on the data before being able to retrieve it, visualise it or perform other interactive exploratory tasks. Such an archive of medium-sized surveys which could be analysed in their entirety using TOPCAT or STILTS could therefore provide a useful new resource for certain kinds of investigation, particularly ones which are not naturally spatially limited. Crossmatching between these survey datasets would obviously be one important use of them.

For the largest surveys local acquisition and interactive investigation of the entire dataset will never be feasible, but use of the techniques and software described here

extend the reasonable size of catalogue which can be investigated in this way considerably higher than has previously been the case.

## 5.5.2 Cached sufficient statistics

When working with very large datasets, running times of algorithms becomes an important issue. One of the main problems with many machine learning and statistical algorithms is that they were developed with the assumption that the data sets they work on are available in random access memory, and so they frequently access the data to make calculations. Because data mining uses data sets which can only be fit in secondary or tertiary storage, the overhead needed to access it makes the problem intractable.

One solution to this problem is to use a process known as cached sufficient statistics, where a data structure is created which summarizes the data being studied. This information is then used to perform some of the calculations without having to access the data directly. The initial cost of building this summary can be expensive, but it only has to be done once and is well worth the time saved in the long run.

One of the most common data structures used are k-d trees, which help with accelerating nearest neighbour searches. k-d trees are binary trees which partition the data-space into axis-aligned boxes. They are constructed recursively, with each node covering a k-dimensional volume which is further partitioned by its children.

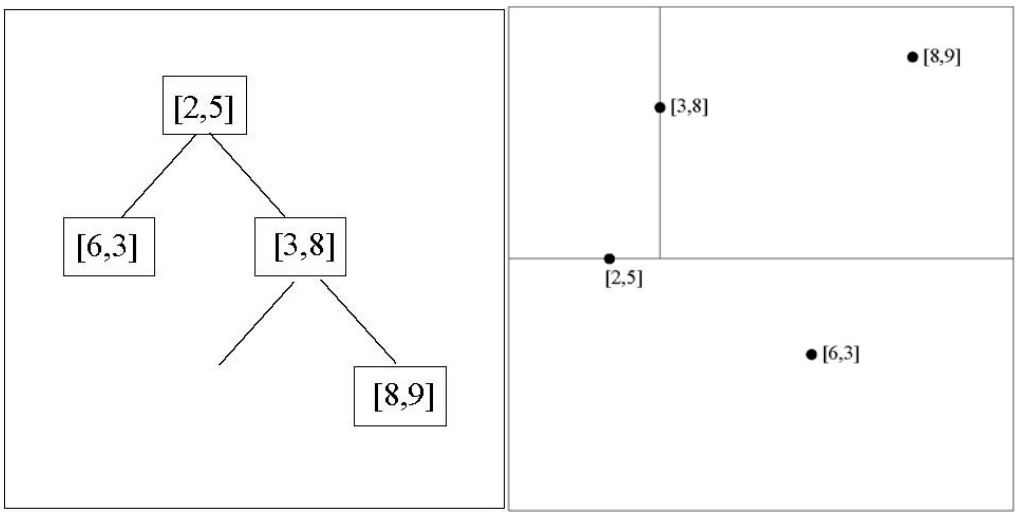


**Figure 39: A k-d tree for four elements**

Figure 39 shows a k-d tree for four elements in a 2D space (left) and, on the right, how the tree partitions the space. These partitions speed up nearest neighbour searches by quickly eliminating points from the search. Only a small subset of points, contained in a few partitions adjacent the point being tested, need to have their distance measured explicitly to find the nearest one.

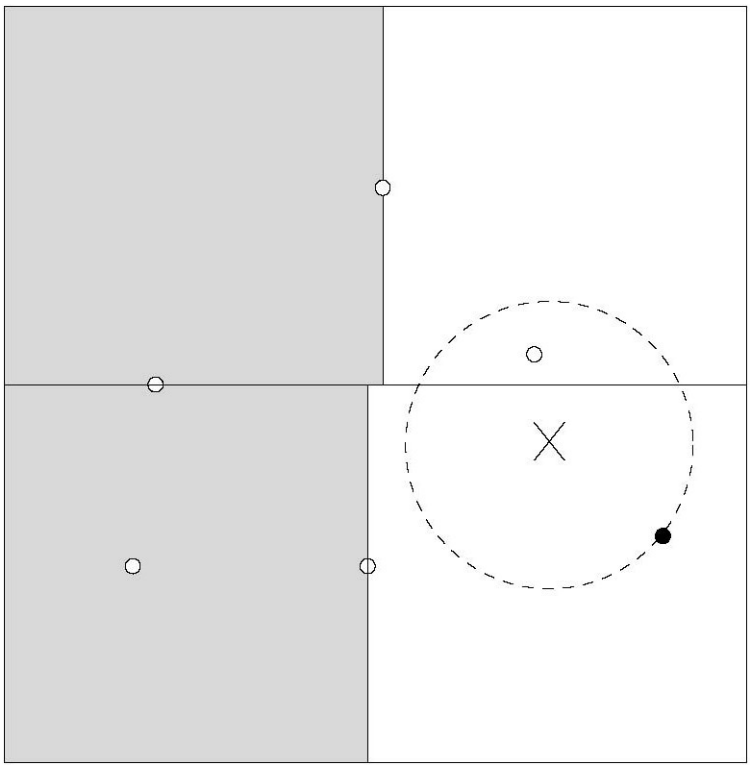

**Figure 40: Using a k-d tree to accelerate nearest neighbour search**

Figure 40 illustrate the use of a k-d tree to accelerate nearest neighbour search. The tree is recursively searched to find the partition that contains the point being tested, (represented by an X), and the nearest neighbour within its bounds (the black dot). This gives an upper bound for the nearest neighbour, any points contained in partitions which do not intersect the dotted circle can be ignored.

k-d tees can be augmented with further information such as the mean and covariance of the points covered by each node to form multi-resolution k-d trees (mrkd-trees[67]). This information can be used to improve the running time of many different algorithms, such as expectation maximization, k-means, kernel density estimation and Bayes Net learning. Often the statistics available in the tree can be used to approximate the results of these algorithms very accurately without any need to access the underlying data.

k-d trees are only effective on data with less than about ten dimensions. This is partially due to the square shape of the area covered by their nodes, and partially due to the way they are constructed. When dealing with higher dimensional data, it is more efficient to use ball trees [68]. Ball trees are similar to k-d trees, only the area covered by their nodes is hyper-spherical. This means that they no longer partition the data-space, and can overlap. Although this makes them more difficult to construct, it does not affect the way they are used. Just like k-d trees they are mainly used to improve nearest neighbour searches, though they can also be augmented with additional information to improve a range of common data mining algorithms, as most algorithms that use k-dtrees can be adapted to use ball trees in their place.

## 5.5.3 Reducing the amount of data to be visualized: Eirik

### 5.5.3.1 Introduction

While sophisticated data structures may help make data mining algorithms more scalable, the problem of visualizing very large datasets is more fundamental: as the size of the dataset increases, it quickly becomes impossible to plot it all – as points or lines or whatever the representation of choice – without exceeding the space on the screen or the brain's capabilities to interpret what is plotted.

In the case of visualization, then, it seems that there is no alternative but to reduce the amount of data to be visualized. Sky survey database tables can have ~$10^2$ columns and ~$10^8$ rows, so techniques which reduce either the number of rows or the number of columns to be plotted can be useful. The following two figures summarise some of these methods:

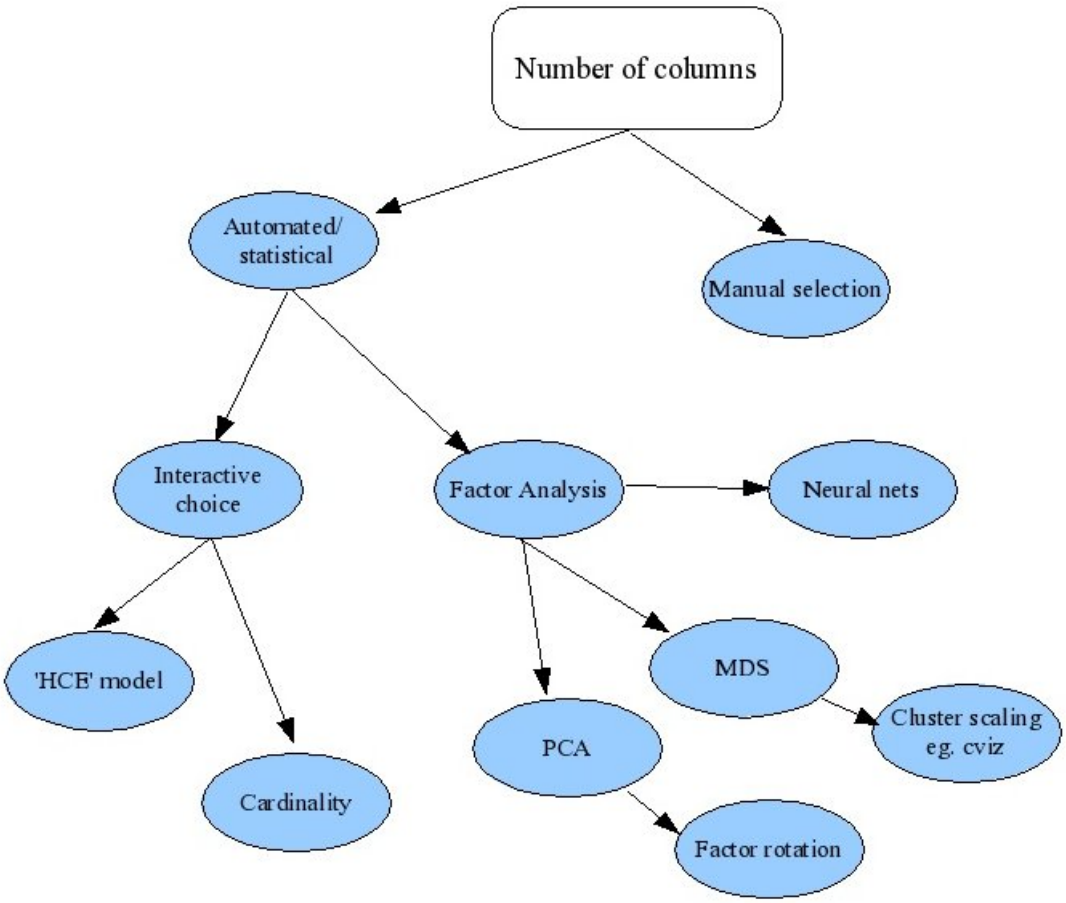


**Figure 41: Some techniques for column reduction**

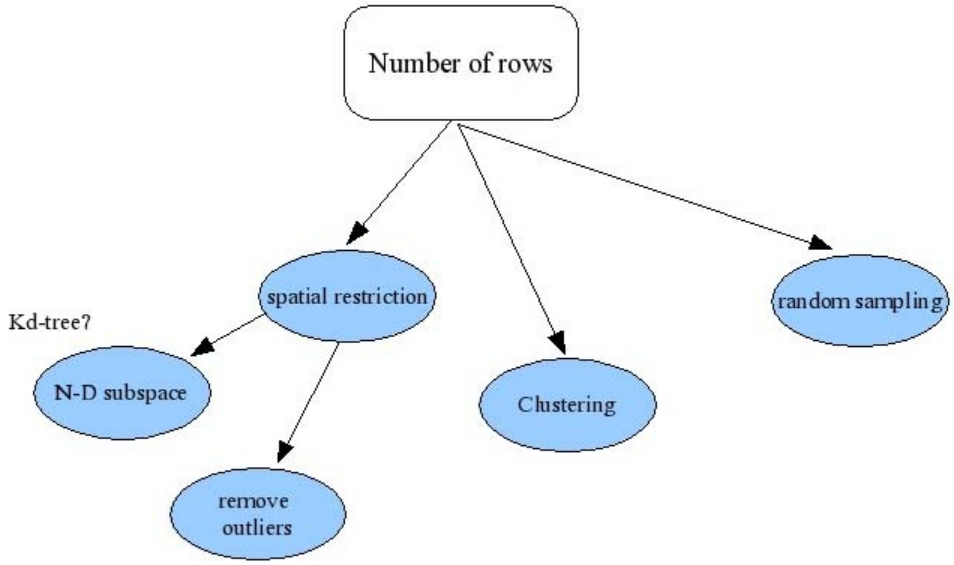


**Figure 42: Some techniques for row reduction**

These two figures are taken from a review of *Dimension Reduction Algorithms for Data Mining and Visualization* [69] undertaken as part of the DS6 work: a fuller summary of that review will be included in the final DS6 *Study Report*, as here we focus on the development of *Eirik*, the prototype tool designed to implement a number of these techniques.

### 5.5.3.2 Design

Eirik is a prototype tool for data navigation; the aim is to provide statistical analyses to support the visualization of large data sets, where, using conventional methods (such as scatterplots [70]), the display may be too saturated to see any clear pattern. Eirik achieves this aim by embedding R [71] (a free software environment for statistical computing and graphics) as a library, which imposes a number of design considerations. In general though, the underlying design of Eirik is governed by three factors:

- As we move into higher dimensions (1D to 2D to 3D etc), the bounding volume enclosing the data grows exponentially, which creates considerable problems in finding and viewing displays of high-dimensional data [88]). It is, therefore, a considerable advantage reduce the number of data points by defining specific regions of interest.
- R provides a powerful scripting interface, which has a distinct learning curve and few security features. It is therefore desirable for knowledgeable users to be able to execute arbitrary scripts, or load particular external libraries, whilst supporting a set of selected functions directly through the Eirik interface. Access to system files can also be curtailed by controlling the remit of a single application (as opposed to, say, sand-boxing - and maintaining - many external R scripts).
- For interactive visualization, there is a recognized requirement to keep update rates down to less than a second (and often, less than 0.1s is necessary). In many cases, it would be sufficient - and far more effective - to display summaries of the data, rather than downloading the data themselves. To this end, Eirik possesses a

simple client-server style interface between the GUI and data processing components, which may readily be mapped to that of ICE[72] or gSoap[73] in future.

These issues are explored further below.

**Selection criteria**

Displaying high-dimensional data in 3D, say, with other dimensions represented by colour, glyph etc can result in a rapidly increasing mental load, which unfortunately means less and less to end users. Thus, there is a great advantage in reducing the number of data points which we are trying to display. Eirik forces the user the make selection decisions before the data are loaded, since only a limited number can be loaded at any one time (the current limit is 10 columns). To speed up this selection process, a binary data format is used and presently, all supported data formats (votable, csv and plain text) are converted to 'votable-fits-href' using a wrapped version of the Stilts library (see Figure 43).

Also, if UCD descriptions are available, these are parsed into a browser 'file-tree' structure, in which uninteresting columns can be easily hidden and others can quickly be selected.

Having loaded the data, there are several options to restrict the range or columns of data for further plotting or analysis:

- The *sampling* dialog allows a subset of the data to be selected, or else an arbitrary mask to be constructed to include/exclude rows of the data table, based on user-specified regions in the other plot windows (below).
- In 1D, the *sparkline* tab, used in conjunction with the plot window, can define regions of interest within the data.
- In 2D, the *levelplot* tab can be used to view correlations between all or part of the data to select suitable pairs of columns, eg for scatterplot axes.

**Embedding R**

Embedding R as a library is currently seen (by R authors) as a rather experimental 'feature'. Instead, R is normally run from a command line interface, which has several pros and cons. An important plus point is that scripts can be written and sourced, allowing complex analyses to be specified and re-run under different conditions or parameters with relative ease. On the downside, the user has to learn the R script language and the fact that this language is so powerful raises a number of security headaches (especially within web service environments).

The Eirik prototype is therefore proposed to highlight these problems and present a workable solution: R code can still be run, or called from the interface, and can be more easily restricted to the user's workspace (as opposed to calling any number of external R scripts). The user's own R scripts can be imported using the R `source` instruction in the *rtab* interface.

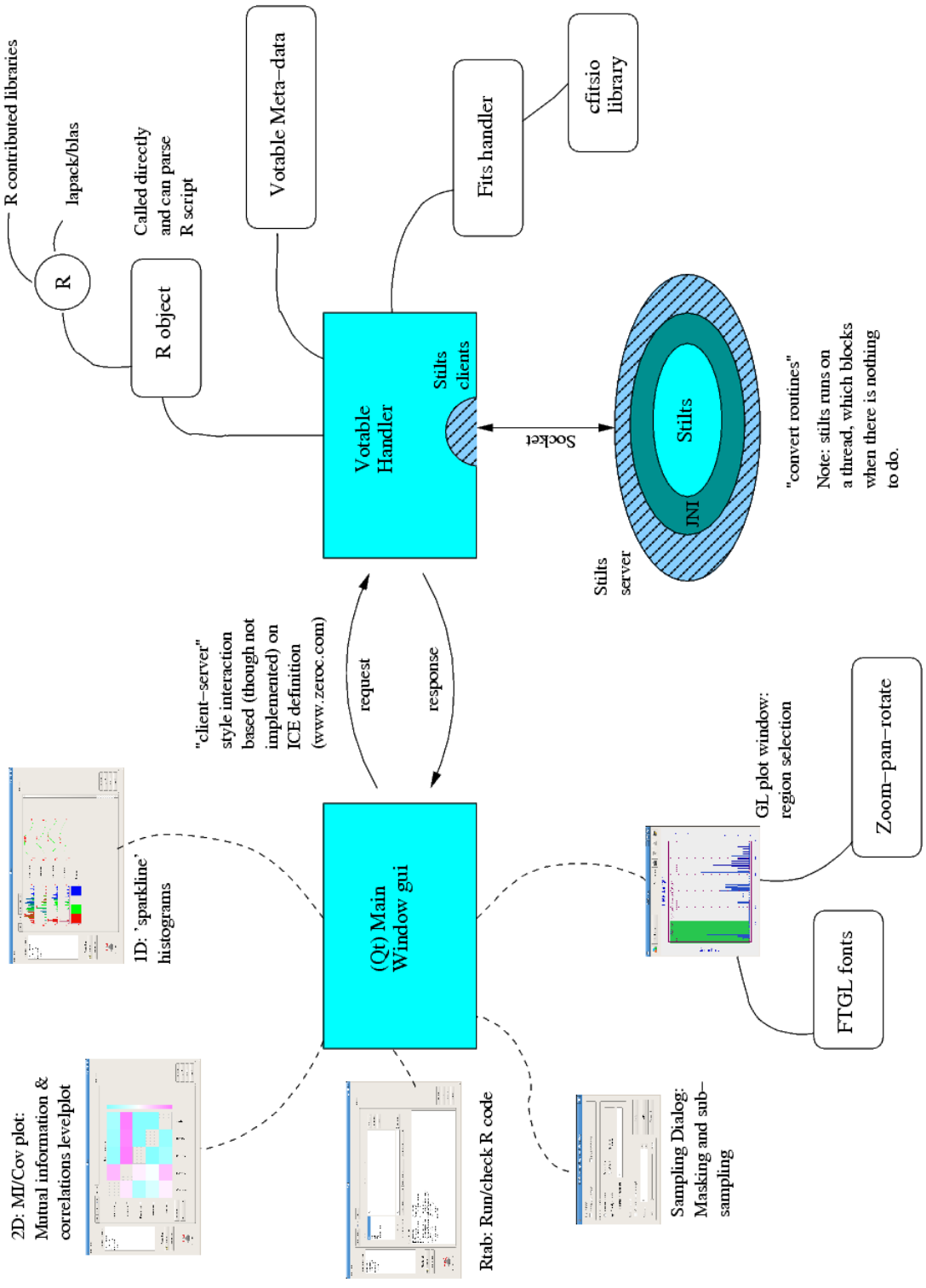


**Figure 43: The client-server data flow in the Eirik design**

## The client-server data flow

Another factor in the design was to permit (at some point in the future) possible client and server versions. A server component might then reside near the data and respond with summaries of the data, rather than gathering (and downloading) all the data. In this way, a user who is only really interested in regions of the data is much more

likely to find clusters or correlations of interest. Whilst this kind of client-server interaction has not been implemented to date, a crucial part of the design is the constraint of data flow between the gui-related and data-handling classes, as illustrated in Figure 43.

In particular, data summaries are requested only by the main GUI window to VOTable handler objects, using an ICE-style specification, which essentially ensures that data in this loop must be serializable.

### 5.5.3.3 Using Eirik

If the tabular dataset loaded into Eirik includes UCDs, then the user is presented with a view like the following:

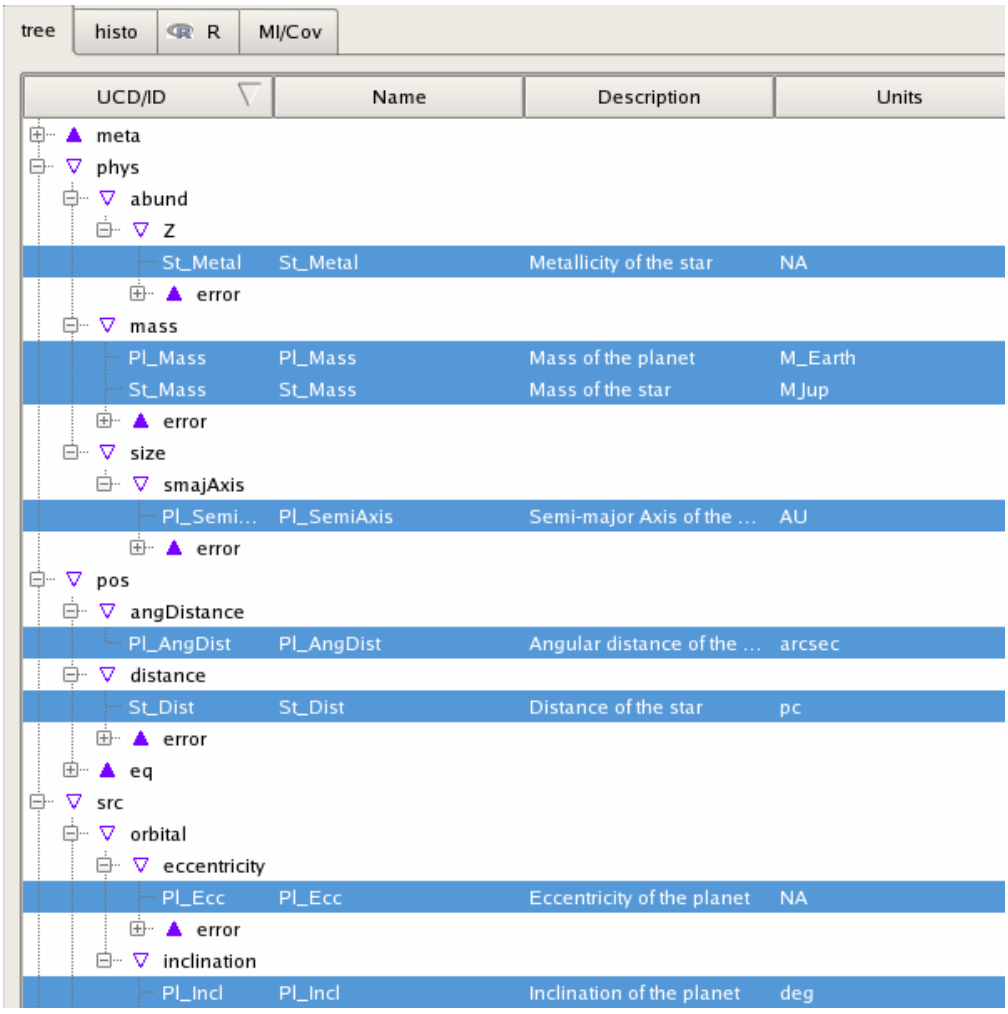


**Figure 44: The UCD tree view in Eirik**

This is a representation of the columns in the table viewed through the UCD hierarchy, and it presents the user with the first way of selecting or removing from consideration certain columns in the dataset. If UCD information is not available, then Eirik presents a simple list of the attributes present in the table, from which selection or rejection can be made in the same way.

Once the data are loaded and a preliminary selection of the attributes of interest have been made, the user can start using Eirik's graphical capabilities.

**1D graphics**

Eirik offers a number of 1D graphical capabilities. For example, in Figure 45, the user has generated a series of five histograms showing the (u-g), (g-r), (r-i) and (i-z) colours of a set of galaxies, plus their redshifts. The user has then selected a range in (u-g) colour and overplotted on the other histograms the location of the sources possessing (u-g) colours within that range. The colour and position of the overplots reveal that (u-g) colour is not a strong predictor of redshift or i-z colour, but that there is a reasonable correlation with both (g-r) and (r-i), albeit with significant scatter, as shown by the mixing of the colours in the overplots.

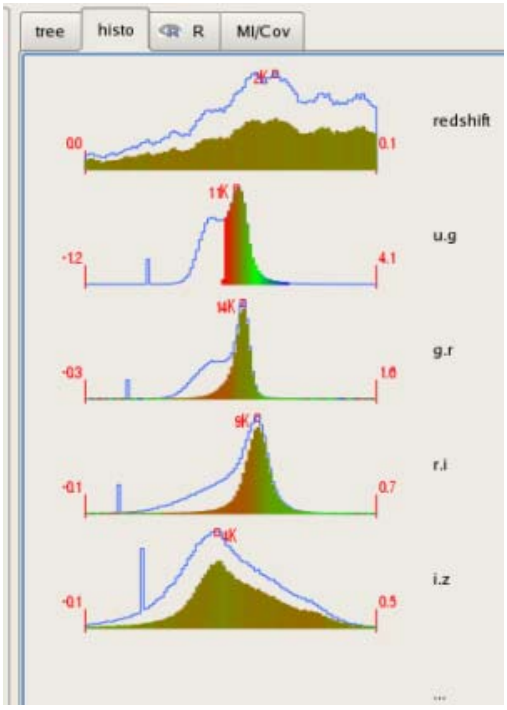


**Figure 45: Histograms in Eirik**

**2D Graphics**

Eirik takes advantage of R's statistics libraries in one of its 2D plots, as shown in Figure 46. This plots the Mutual Information and Covariance values for pairs of attributes in a dataset. Mutual Information and Covariance are complementary statistics for determining relationships between attributes, so this plot can be used either to discover pairs of attributes with significant, and, perhaps, hitherto unknown, relationships or for identifying pairs of attributes which are so strongly related that they do not both need to be considered in the data exploration analysis at hand.

## 5.5.3.4 Summary and Conclusions

The Eirik prototype has implemented only a fraction of the techniques which could be used in the reduction of data volume through the selection of interesting combinations of rows or columns, but it has yielded a number of useful lessons. The availability of a rich statistical library like R is clearly very beneficial, but details of its

implementation (e.g. the ease of making system-level calls from R scripts, and the fragility of the experimental R servers available) means that it would not be easy simply to wrap R as a web service in what would be the natural VO manner without causing security issues. What does seem clear is that something like the functionality prototyped in Eirik would be very valuable within the VO.

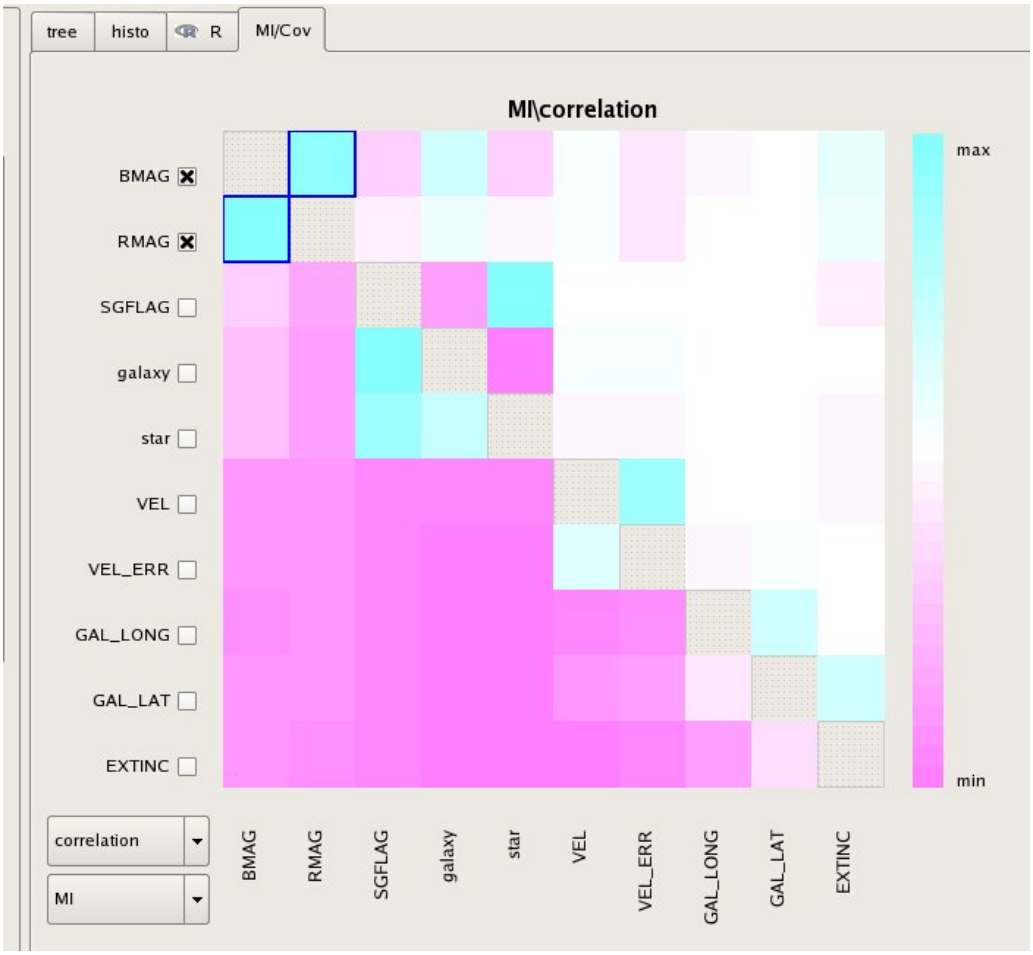


**Figure 46: A Mutual Information/Covariance plot in Eirik**

## 5.5.4 The VOTechBroker: parameter sweeps on the Grid

### 5.5.4.1 Overview

The Virtual Observatory aims to provide mechanisms for astronomers to use globally distributed services in a transparent and seamless manner for data mining and visualisation. One type of common type of service is the parameter sweep, whereby the same algorithm is executed potentially thousands of times, on the same data set, but with partially differing input parameters, in order to discover (or mine) new knowledge. Parameter sweeps can require a large amount of computational power, furthermore individual process execution times within a parameter sweep can vary by orders of magnitude, in response to a change in a single input parameter value. In order to minimise the latency associated with obtaining results scientists often demand larger amounts of computational power than their local instituitions can

provide. In the field of computer science, computational grids [74] have been proposed as a way of sharing non-trivial quality of service for computing resources between members of a Virtual Organisation. However, the exact standards and middleware that comprise a Grid have changed a great deal over the past few years, leading to a plethora of standards and software implementations that may be available at a site donating resources. The VOTechBroker (VOTB) [75] has been created to provide a bridge between AstroGrid's implementation of the VO framework and existing computational resources. In particular the VOTB aims to be middleware-agnostic so that new algorithms and types of computational resource can be provided through the AstroGrid middleware in a transparent way.

### 5.5.4.2 Motivation

The motivation for the VOTB was driven by a number of astronomy use cases and algorithms, for example a 3-point correlation function for measuring the spatial distribution of galaxies in the universe. On a practical level the VOTB was also motivated by the desire to obtain as many computational resources as possible from partner organisations, in order to speed up the results obtained from the 3-point function. To help achieve this goal, we determined that a partner organisation was more likely to donate their resources to the Virtual Organisation if the VOTB's software requirements at each site meant that the site had to perform little or no installation/configuration of their local resources. This implied that we either:

1. Chose a popular grid middleware and ignored sites that either did not already provide that particular middleware, or could not be persuaded to install it.
2. Or, we embraced middleware heterogeneity *within* the VOTB.

At the time of starting this work, the state of the art brokers (e.g. Gridway [76]) all focused on a particular middleware (e.g. a particular version of the Globus Toolkit [77]). A key aim of the VOTB was to support an extensible range of (grid) middleware, hide heterogeneity, and ease the complexity associated with job submission/execution. Furthermore it was desirable to allow each client to submit a computation, disconnect their laptop from AstroGrid, and later reconnect to determine the status of the computation and retrieve output. To achieve this, the VOTB was designed to provide abstract mechanisms to perform the following actions in a heterogeneous resource environment:

1. Describing job and resource requirements,
2. Describing and locating suitable resources (e.g. based on user credentials and CPU architecture),
3. Submitting job requests,
4. Staging data, executable and support files,
5. Retrieving status and result files.

As well as hiding heterogeneity across different sites, VOTB clients (human or computer) would also not be exposed to changes as sites upgraded or changed their middleware.

Another aim was that VOTB should be generic enough so that different astronomers (or their support technician) could take their own particular algorithms and submit them to Grid, from AstroGrid, with minimal effort. Finally, the submission of jobs to

Grid resources should be achieved seamlessly using the existing AstroGrid infrastructure.

### 5.5.4.3 Design

Figure 47 shows the VOTB's logical design, which comprises of three layers.

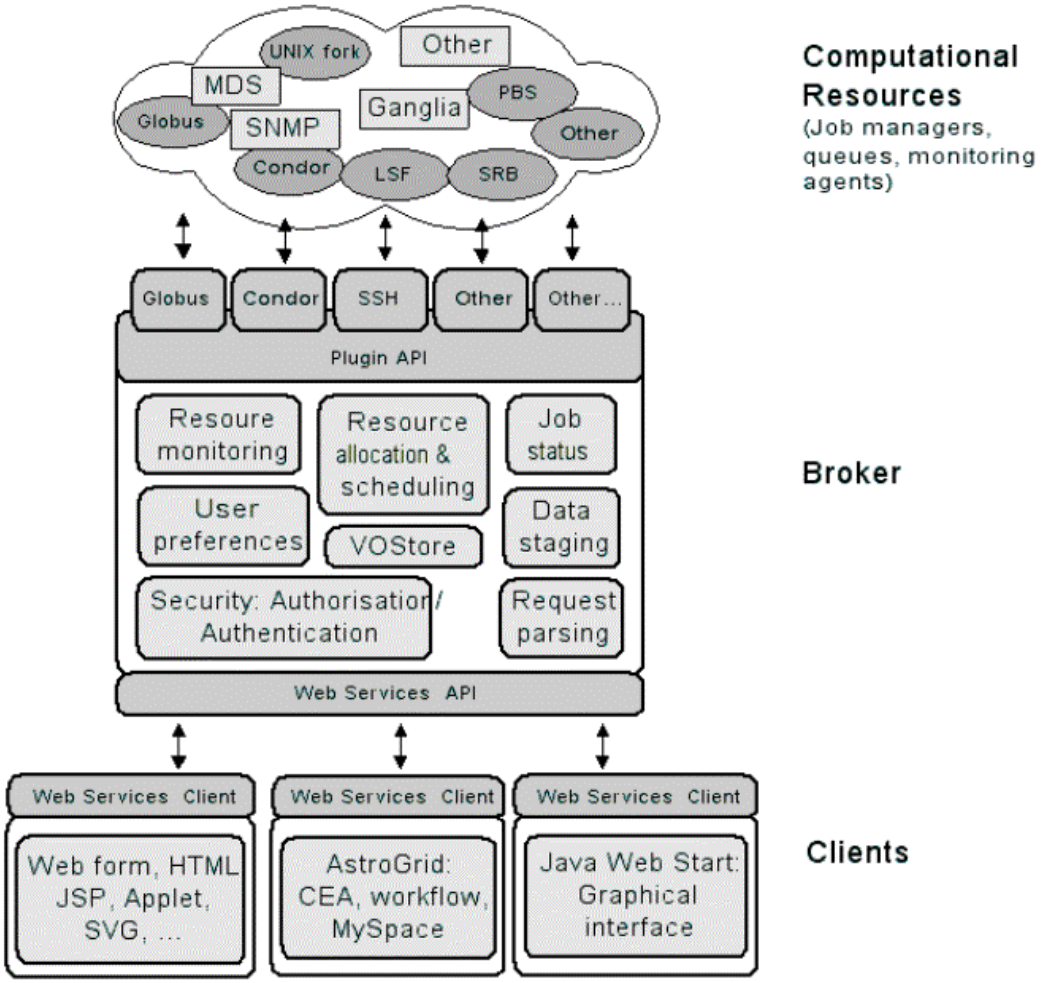


**Figure 47: VOTB: Logical architecture**

Starting at the bottom of Figure 47:

- **Bottom layer**: Clients interact with the VOTB over a Web Services (or similar) API. Multiple types of client can exist to exploit different deployment scenarios. For example a processing element within an AstroGrid workflow, or a graphics client that directly renders the results from a job submission. Clients are responsible for describing their particular job requirements in a common and abstract language, which they communicate to the broker. As long as a client can access the middle layer over the network, it can be located anywhere (e.g. on an 'official' AstroGrid machine, or at a client's site.

- **Middle Layer**: This layer contains the core broker components that provide the services required for submitting job requests, staging binaries and data files, monitoring progress and retrieving results. The functionality here is generic in that any jobs described in the abstract notation are treated in a common way. The plug-in API is the boundary where the abstract notation is converted into resource-specific protocols and syntax. The VOTB locates appropriate resources based on hints provided by the client that describe the submitted algorithm's requirements. The mapping of the abstract notation into the appropriate protocol for a particular underlying middleware is hidden within the associated plug-in. Therefore, to support a range of different middleware; a driver must exist for each one. The client is oblivious to the issue of installed drivers and middleware versions, except for the fact that given a larger number of drivers, potentially a much larger number of resources can be utilised for executing the client's parameter sweep.
- **Top Laye**r: Computational resources existing at remote sites. Interaction is via the plug-in API. Resources include job managers, storage and information services. Resources can come and go as Virtual Organisations are created to satisfy a particular need and then torn down afterwards. Therefore it is essential that resources can be found dynamically based on their current status and availability.

### 5.5.4.4 Implementation

Figure 48 shows how version 1 of the VOTB has been implemented.

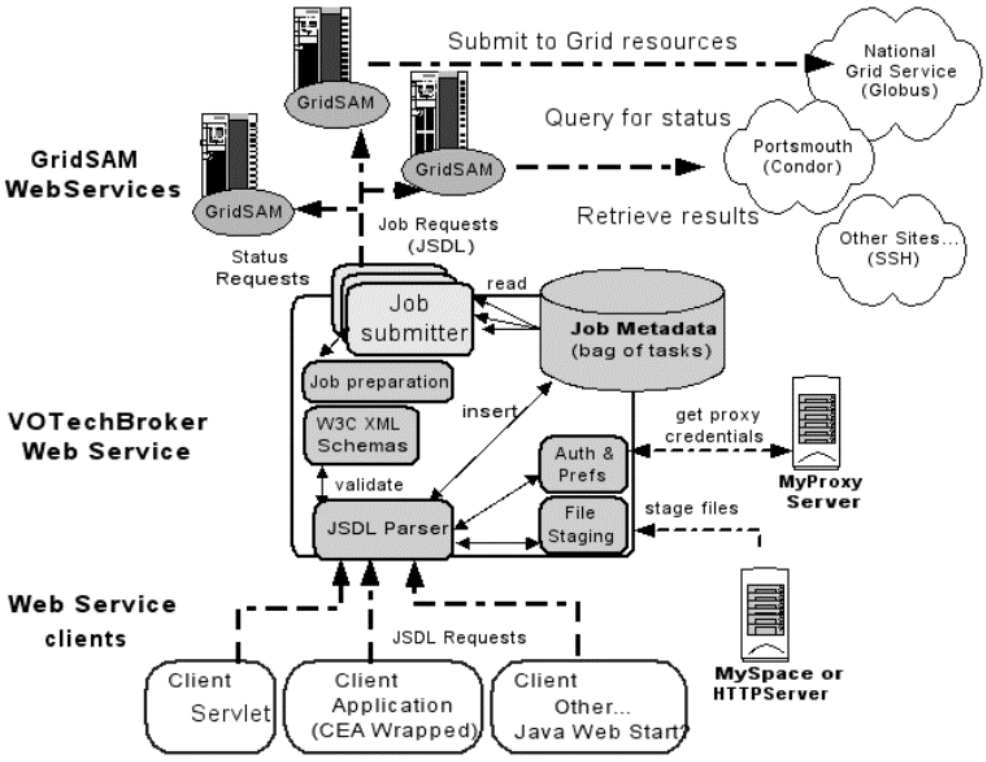


**Figure 48: VOTB implementation**

The Job Description and Submission Language (JSDL[78]) from the Global Grid Forum (GGF[79]) has been selected as the abstract job description language. Authentication is provided using X.509 certificates and myproxy[80] credentials; clients embed myproxy credential metadata within their JSDL requests. The VOTB Java Web Service extracts these metatdata to locate the client's myproxy credential which the VOTB then uses to determine if: (i) the client is permitted to submit jobs via the VOTB; and (ii) which computational resources the client is authorised to use.

The JSDL also describes file staging data, so that binaries and data files can be staged across to the computational resources and results retrieved from across the range of resources involved in the parameter sweep to a particular location specified by the client. To date we have primarily been using http and sftp to stage data. MySpace support was investigated, but put on hold until a number of technical issues could be resolved relating to MySpace interaction.

Parameter sweep definitions are currently broken down into their consistuent parts and placed in a JavaSpace [81] bag of tasks, where a number of submission workers locate appropriate resources for a given job, gather an appropriate number of tasks for execution (e.g. a cluster job manager might accept 100 processes at a time, whilst a uni processor machine accessed by fork over SSH may only accept 1 or 2 processes at a time). The use of the JavaSpace means that submission workers can be distributed over the VOTB's local area network (or system area network in the case of a cluster) in order to provide scalability for large parameter sweeps. See Figure 49.

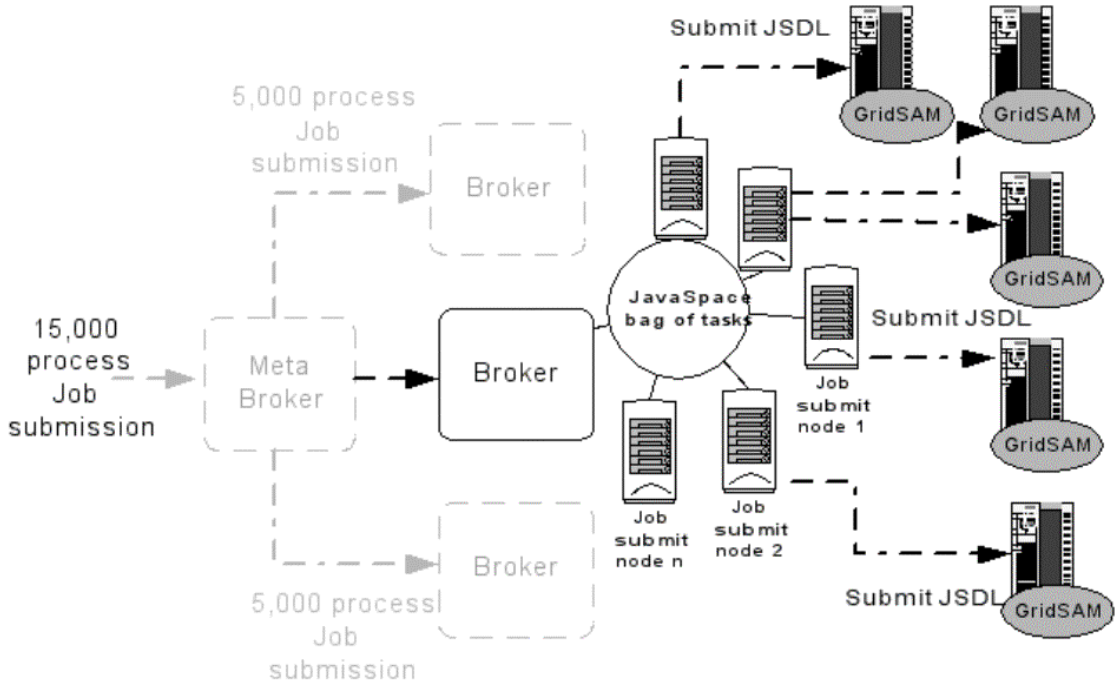


**Figure 49: JavaSpace and the distribution of submission workers over a LAN or SAN**

Version 1 of the VOTB is partly based on the GridSAM [82] system from the London e-Science Centre. GridSAM provides the plug-in architecture that we use to interact with different middleware. Submission workers use the Tycho [83] messaging system to search for available GridSAM instances in a distributed registry, and choose a

suitable instance based on the metatdata provided in the JSDL and the GridSAM's service description held in the registry.

Currently plug-ins for Globus 2.4.3, Condor [84], secure shell, and UNIX fork have all been used to submit to the UK National Grid Service [85], a Condor pool, and symmetric multiprocessor machines. The ability to sumbit jobs to fork over SSH means that resources that do not have a grid middleware installed can used seamlessly with their grid-enabled peers.

#### 5.5.4.5 Algorithms

To date we have primarily been focusing on an N-Point correlation code to test our assumptions about the broker. However we have also been experimenting with the kdtree, ntropy and swarp algorithms. We have successfully submitted N-point parameter sweeps comprising of many hundreds of tasks, the results from which are currently being analysed by a VOTECH astronomer.

#### 5.5.4.6 Summary

The VOTB is a broker to submit parameter sweeps to the Grid, and other distributed resources, in a transparent way. The VOTB aims to interoperate with a wide range of job submission systems using a plug-in system, and to protect the astronomer from middleware details. The plug-in approach has been chosen to maximise the number of computational resources that are available to process a job submission. It is intended that the VOTB provides a bridge between the AstroGrid and Grid resources by appearing as a wrapped component in the AstroGrid workflow environment. In this first prototype we have aimed to be standards-compliant, and have successfully submitted parameter sweeps comprising of many hundreds of tasks. A version 2 of the VOTB is planned which will address a number of issues that were identified in version 1, for example the verbosity of the JSDL notation. Further information can be found on the VOTB Website at: https://portals.rdg.ac.uk/votb.

# 6. Conclusions and Future Work

The DS6 work to date has demonstrated the need for data exploration tools which operate on both the client- and the server-side. There exist a rich variety of client-side tools – many mature products, delivering a wide range of functionality, and developed both within and outwith the astronomical community – so the goal for DS6 must be to find a straightforward way to make them interoperable (with PLASTIC or something similar) and to lower the barrier for the use of further tools within the VO infrastructure. The experience with Weka and VisIVO shows that this is a realisable goal, especially when software like the Astro Runtime can do much of the work for the application developer. The DS6 team developed a Wiki page on how to VO-enable data exploration tools, and a task for the VO community in the longer run is to keep such information updated in the light of developing standards.

On the server side, techniques exist to improve performance: DS6 has started investigating data structure and data storage techniques, and will continue to do so for

the rest of the project. The client and server sides are bridged by a tool like Eirik, which uses metadata (and then data) analysis to restrict the quantity of data to visualize. The VOTECHBroker shows how the VO can be coupled to computational Grid resources when required for analyses which cannot be further reduced in scale.

The challenge for the DS6 team is to develop ways of making all the techniques discussed in this report available for use by the average astronomer. In many cases doing that requires interaction with the data centres whose servers must run the necessary software, so DS6 must impress upon the DCA the importance of data exploration for the future of the EuroVO.

# References


1. European Virtual Observatory (EuroVO): http://www.euro-vo.org/pub/
2. International Virtual Observatory Alliance (IVOA): http://www.ivoa.net
3. IVOA Applications Working Group: http://www.ivoa.net/twiki/bin/view/IVOA/IvoaApplications
4. OPTICON: http://www.astro-opticon.org/
5. RadioNet: http://www.radionet-eu.org/
6. Enabling Grids for e-Science (EGEE): http://public.eu-egee.org/
7. EuroVO Data Centre Alliance: http://www.euro-vo.org/pub/dca/
8. VOTable specification: http://www.ivoa.net/Documents/latest/VOT.html
9. Wikipedia entry for k-d tree: http://en.wikipedia.org/wiki/K-d_tree
10. Aladin: http://aladin.u-strasbg.fr/
11. TOPCAT: http://www.star.bristol.ac.uk/~mbt/topcat/
12. GAIA: http://star-www.dur.ac.uk/~pdraper/gaia/gaia.html
13. DS9: http://hea-www.harvard.edu/saord/ds9/
14. PLASTIC website: http://PLASTIC.sourceforge.net
15. Hand D.J., Mannila H., Smyth P., 2001, *Principles of Data Mining* (MIT Press)
16. Andrew Moore's Statistical Data Mining tutorials: http://www.autonlab.org/tutorials/
17. Witten I.H., Frank E., 2005, Data Mining: Practical Machine Learning Tools and Techniques (Morgan Kaufmann)
18. Weka: http://www.cs.waikato.ac.nz/~ml/index.html
19. Sloan Digital Sky Survey (SDSS): http://www.sdss.org
20. Richards G.T., et al, 2002, AJ, 123, 2945. *Spectroscopic Target Selection in the Sloan Digital Sky Survey : The Quasar Sample*
21. Richards G.T., et al, 2004, ApJS, 155, 257. *Efficient photometric selection of quasars from the Sloan Digital Sky Survey: 100,000 z < 3 quasars from Data Release 1*
22. Astroneural: http://people.na.infn.it/~astroneural/index.html
23. D'Abrusco R., Staiano A., Longo G., Brescia M., Paolillo M., De Filippis E., Tagliaferri R., 2007, ApJ, submitted. *Mining the SDSS archive. I. Photometric redshifts in the nearby universe*
24. FITS: http://fits.gsfc.nasa.gov
25. NRAO VLA Sky Survey (NVSS): http://www.cv.nrao.edu/nvss/
26. STILTS: http://www.star.bristol.ac.uk/~mbt/stilts/

27. (Initial) DS6 Software Survey: http://wiki.eurovotech.org/twiki/bin/view/VOTech/DS6SoftwareSurvey
28. Mirage: http://cm.bell-labs.com/who/tkh/mirage/index.html
29. xmdvtool: http://davis.wpi.edu/xmdv/
30. VOPlot: http://vo.iucaa.ernet.in/~voi/voplot.htm
31. Octet: http://www1.cadc-ccda.hia-iha.nrc-cnrc.gc.ca/cvoProto/
32. Fernique, P., 2006 VOApplication Interoperability Study http://www.ivoa.net/internal/IVOA/InterOpMay2006Applications/Victoria2006.ppt
33. VisIVO: http://visivo.cineca.it/
34. SC4DEVO: http://www.roe.ac.uk/~rgm/sc4devo/
35. Winstanley N. et al., 2006, in Astronomical Data Analysis Software and Systems XVI ASP Conference Series. *Astro Runtime - Client-Side Middleware for the VO*
36. XML-RPC: http://www.xmlrpc.com/
37. Digby A. P., Hambly N.C., Cooke J.A., Reid I.N., Cannon R.D.,2003, MNRAS, 344, 583. *The subdwarf luminosity function*
38. SuperCOSMOS Science Archive (SSA): http://surveys.roe.ac.uk/ssa
39. IVOA Note on PLASTIC: http://ivoa.net/Documents/latest/PLASTICDesktopInterop.html
40. IVOA Identifiers: http://www.ivoa.net/Documents/latest/IDs.html
41. AstroMD: http://www.cineca.it/sap/area/cosmolab.htm
42. Unified Content Descriptors (UCDs) : http://www.ivoa.net/Documents/latest/UCD.html
43. MySpace : http://www2.astrogrid.org/science/documentation/workbench-system-services/myspace/myspace-help
44. VOSpace : http://www.ivoa.net/twiki/bin/view/IVOA/VOSpaceHome
45. Storage Resource Broker (SRB): http://www.sdsc.edu/srb/index.php/Main_Page
46. PlasKit: http://www.star.bristol.ac.uk/~mbt/PLASTIC/
47. Simple Image Access Protocol (SIAP): http://www.ivoa.net/Documents/latest/SIA.html
48. Simple Spectral Access Protocol (SSAP): http://www.ivoa.net/twiki/bin/view/IVOA/SsaInterface
49. AstroScope: http://www2.astrogrid.org/science/documentation/data-explore/astroscope
50. Ochsenbein F., Bauer P., Marcout J., 2000, A&AS 143, 221
51. INAF: http://www.inaf.it/
52. CINECA: http://www.cineca.it/
53. Viceconti M., Astolfi L., Leardini A., et al. 2004, IEEE Proceedings of IV2004. *The Multimod Application Framework*
54. APT: http://www.stsci.edu/hst/proposing/apt
55. VOSpec: http://esavo.esa.int/vospec/
56. Specview : http://www.stsci.edu/resources/software_hardware/specview
57. eSTAR: http://www.estar.org.uk/wiki/index.php/Main_Page
58. http://wiki.eurovotech.org/twiki/bin/view/VOTech/TopcatAladinSnapshots
59. Louys M., Bonnarel F., Ochenbein F., Fernique P., Genova F., Murtagh F., Wenger M., Didelon P., 2004. in Toward an International Virtual Observatory: Proceedings of the ESO/ESA/NASA/NSF Conference Held at Garching, Germany, 10-14 June 2002, ESO ASTROPHYSICS SYMPOSIA. ISBN 3-540-21001-6. Edited by P.J. Quinn and K.M. Górski. Springer-Verlag Berlin/Heidelberg, 2004, p. 292

60. STIL: http://www.star.bris.ac.uk/~mbt/stil/
61. MIDAS: http://www.eso.org/projects/esomidas/
62. STSDAS: http://www.stsci.edu/resources/software_hardware/stsdas
63. HDS: http://www.starlink.rl.ac.uk/star/docs/sun92.htx/sun92.html
64. Sybase-IQ: http://www.sybase.com/products/datawarehousing/sybaseiq
65. MySQL: http://www.mysql.com/
66. 2MASS: http://www.ipac.caltech.edu/2mass/
67. Moore A., 1999, in M. Kearns and D. Cohn (eds), Advances in Neural Information Processing Systems, p. 543. *Very Fast EM-based Mixture Model Clustering Using Multiresolution KD-trees*
68. Moore A., 2000, in Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence, p. 397. *The Anchors Hierarchy: Using the Triangle Inequality to Survive High-Dimensional Data*
69. Holbrey R., http://www.comp.leeds.ac.uk/richardh/astro/pdf/alg1.pdf, *Dimension Reduction Algorithms for Data Mining and Visualization*
70. E Bertini & G Santucci, 2006, Give chance a chance: modeling density to enhance scatter plot quality through random data sampling, Information Visualization (2006) 5, 95-110.
71. R: http://www.r-project.org/
72. ICE: http://www.zeroc.com/ice.html
73. gSOAP: http://gsoap2.sourceforge.net/
74. Foster I., and Kesselman C., (eds.), 2003, *The Grid2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann
75. The VOTechBroker: https://portals.rdg.ac.uk/votb
76. GridWay: http://www.gridway.org/
77. The globus alliance: http://www.globus.org/.
78. JSDL: https://forge.gridforum.org/projects/jsdl-wg
79. Global grid forum: http://www.ggf.org/
80. MyProxy Online Credential Repository: http://grid.ncsa.uiuc.edu/myproxy
81. Java Space: http://www.sun.com/software/jini/
82. GridSAM: http://www.lesc.ic.ac.uk/gridsam
83. Tycho: http://www.acet.rdg.ac.uk/projects/tycho/index.php
84. Condor: http://www.cs.wisc.edu/condor/
85. National Grid Service: http://www.ngs.ac.uk
86. Java-RMILite: http://sourceforge.net/projects/rmi-lite
87. PLASTIC applications: http://plastic.sourceforge.net/apps.html
88. DW Scott, 1992, *Multivariate density estimation: theory, practice, and visualization.* Wiley Publications.