Astronomical Data Analysis Software and Systems VII
ASP Conference Series, Vol. 145, 1998
R. Albrecht, R. N. Hook and H. A. Bushouse, eds.

Using Java for Astronomy: The Virtual Radio Interferometer Example

N.P.F. McKay[1] and D.J. McKay[1]

Nuffield Radio Astronomy Laboratories, University of Manchester, Jodrell Bank, Macclesfield, Cheshire SK11 9DL, UK

Abstract. We present the Virtual Radio Interferometer, a Java applet which allows the demonstration of earth-rotation aperture synthesis by simulating existing and imaginary telescopes. It may be used as an educational tool for teaching interferometry as well as a utility for observers. The use of the Java language is discussed, and the role of the language, within the astronomical software context, reviewed.

1. Introduction

In 1996, the Australia Telescope National Facility (ATNF) was awarded a grant to upgrade its existing interferometer — the Australia Telescope Compact Array (ATCA). The ATCA is a 6 kilometre earth-rotation aperture synthesis telescope, consisting of six movable 22 metre antennas located on an east-west railway track (JEEEA 1992). Each antenna can be moved and placed at various locations along this track, known as stations. Part of the upgrade involves the construction of new stations on the existing east-west track and the addition of a north-south track. These extra antenna positions will improve the imaging capability of the instrument.

To help astronomers and engineers at the various institutions experiment with and visualise the effects of different positions of the new stations, a design and simulation software package was required. This program had to be intuitive to use, widely available and quickly developed. The resulting program was called the Virtual Radio Interferometer[2] (VRI).

2. The Virtual Radio Interferometer

The idea behind the VRI project was to create an interactive, user configurable, simulated interferometer, providing a tool for determining the synthesised aperture of real and hypothetical telescopes. It fosters an intuitive feel for antenna positions, uv-coverage and the resultant point-spread function. VRI also proved

[1]Formerly: Australia Telescope National Facility, CSIRO, Paul Wild Observatory, Locked Bag 194, Narrabri NSW 2390, Australia

[2]http://www.jb.man.ac.uk/vri/

to be a useful teaching tool, in addition to its original design role of investigating new antenna arrays and experimenting with "what-if" scenarios.

As well as exploring the properties of several existing instruments, the user has control of a number of telescope parameters, allowing the creation of hypothetical interferometers. These include the latitude of the site and the number of array elements which form the interferometer, together with their diameter, elevation limit and position on a two dimensional plane. The declination of the source, as well as the frequency and bandwidth of observation can also be specified. Once these parameters are set, a corresponding *uv-plot* is produced by the program, showing the aperture-plane coverage.

Given the plot of the uv-coverage, we can simulate its effect on various objects. An image of the source is loaded onto one of four display panels, a Fourier transform performed on it, and also displayed. This can be then masked with the uv-coverage. When the inverse Fourier transform is applied, the original image of the source is replaced by the image obtained by "observing" it with the simulated interferometer. Finally, uv-plots generated by a number of antenna configurations, or even different observatories, may be combined to generate an image composed of multiple "observations".

3. Implementation

The development path of the project, and hence the choice of programming language, was governed by several constraints. Firstly, the program had to be easy to use and have a short learning curve. It was decided that VRI would be interactive, present a predominantly graphical interface and be mouse-driven. To encourage people to use the it, the application had to be easily accessible, so the World Wide Web was chosen as the transport medium and, as we had no control over the platform that our audience was using, the code had to be portable. Above all, rapid development was required.

Java was chosen as the coding language, because it satisfied these requirements. As a language, it is high-level, strictly object-oriented, and easy to learn. Classes which facilitate the writing of software for Internet distribution form part of the standard Java Development Kit. Hence, setting a program up to run on the Web is not a complex exercise. The source code is partially compiled into a set of *byte codes* before released for use. These are platform independent, hence eliminating the need for multiple versions of the software. The byte code is downloaded onto the client computer and, at run time, is interpreted into native instructions and executed by the host machine. This alleviates congestion on the organisation's HTTP server, and once the program is downloaded onto a local machine, provides a faster program for users.

4. Appraisal of the Project

On the whole, Java proved to be an ideal development language for this particular project. This is not to say that no problems were encountered.

Being a semi-interpreted language, programs run slower than their native code counterparts. For example, a 256×256 point Fourier transform takes 15 seconds to be executed by the Java interpreter, on a Sun Ultra-1 running at

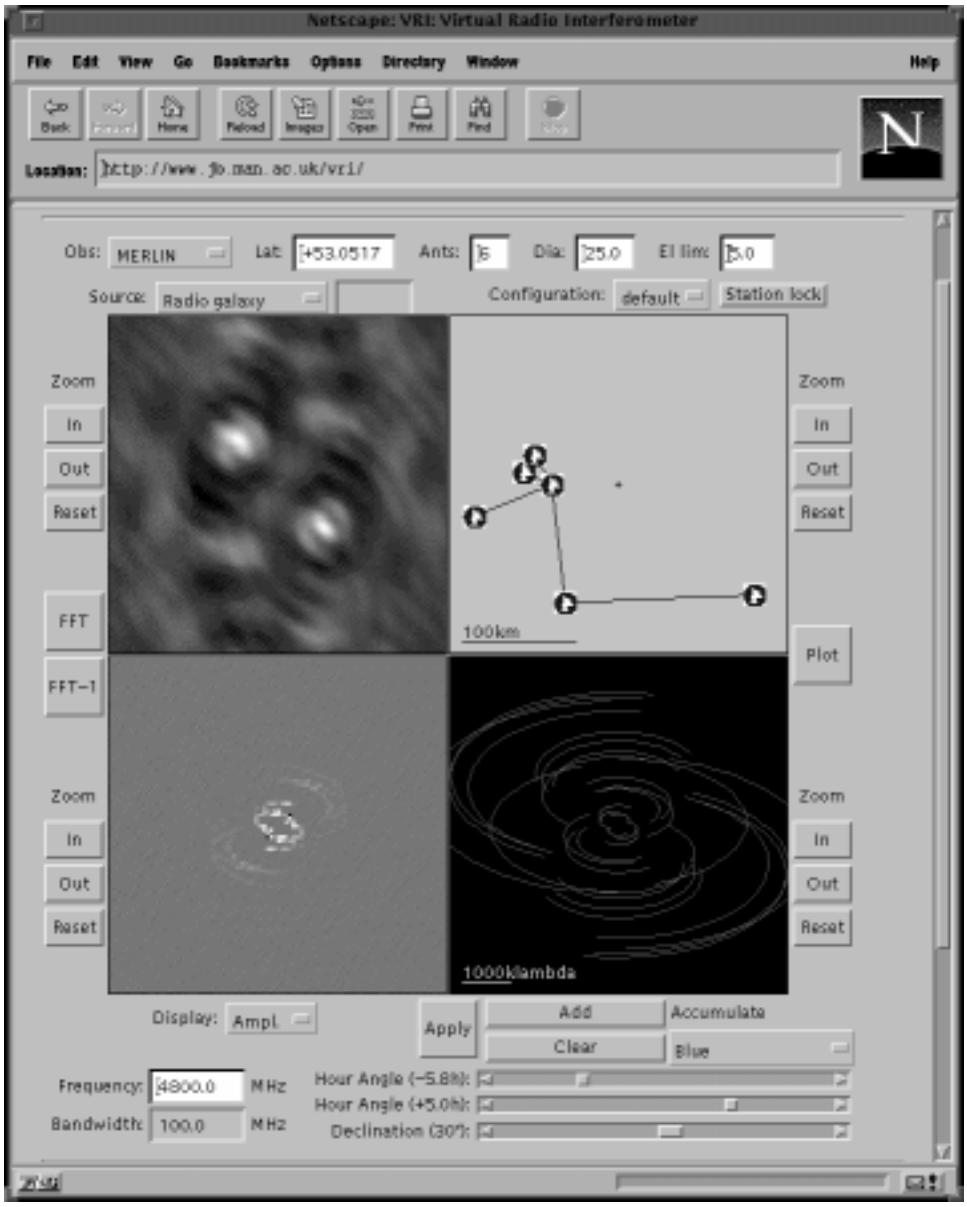

Figure 1. The VRI Java applet, run from a Web browser.

140 MHz, which can be compared with figures of less than one second for native code. Other potential reasons for avoiding the language include the fact that Java is still somewhat immature and evolving, the lack of complete control over registers, memory, etc., provided by other languages such as C and C++ and the problems posed by integrating a new language with legacy code.

Nevertheless, the portability of the language, the rich vocabulary of standard class functions, as well as Java's easily extended object-oriented design, greatly outweigh these disadvantages.

Table 1. Software development metrics for VRI

Java Development Kit Version	1.0.2	
Software classes	26	
Java instructions	1565	
Development time	80	hours
Size of byte code	1.3	Mbyte

5. Future Development

VRI has proved to be a successful software project and users have provided favourable feedback. Improvements have been planned, and subsequent versions of VRI will include deconvolution, three-dimensional arrays and the ability to import real astronomical data. As Java has demonstrated itself to be a model environment for the coding of this application, it will continue to be utilised.

The authors are of the opinion that the Java programming language has a useful role to play in general astronomical software. We would not, however, recommend its use for all applications. Its strength lies in the regime of off-line applications, interactive tools and, of course, user interfaces.

Acknowledgments. The authors would like to thank Mark Wieringa for his contribution to this project, as well as Andrew Lyne, Jim Caswell, John Reynolds and Dave Shone.

The Australia Telescope is funded by the Commonwealth of Australia for operation as a National Facility managed by CSIRO.

References

June 1992, JEEEA (Journal of Electrical and Electronics Engineering, Australia), 12, 2