A Comparative Study of Two Astronomical Software Packages

Ching Yi Tsai

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science,

The University of Auckland, 1999

To Dad, Mum, and Chia Chin

Abstract

This thesis is a comparative study of two open source astronomical software packages widely used by professional observational astronomers - IRAF by USA's National Optical Astronomy Observatories (NOAO) and MIDAS by the European Southern Observatory (ESO).

We evaluate various aspects of the two software packages - software architecture, application programming environment, graphical user interface, system requirements and installation. A comparative experiment provides an objective evaluation of the two software packages.

We also investigate the current status and future plans for the two software packages, and discuss briefly three of the alternative open source and commercial astronomical software packages available on the market - AIPS, AIPS++ and IDL.

Finally, this thesis makes the claim that IRAF is a feature-packed astronomical software that is difficult to install, and has an unintuitive user interface. In contrast, MIDAS is a professional astronomical software that is simple to install and easy to use. To this end, this thesis recommends MIDAS to prospective users.

Acknowledgements

I would like to thank my supervisor, Professor Clark Thomborson. Not only has he given me expert advice in the field of computing, he has taught me how to write a good thesis and become a serious researcher.

Professor Phil Yock has introduced me to real research in astronomy, and has given me the opportunity to present a paper in PEP6. Thank you very much!

Paul Warhurst deserves special mention. He is a good friend and a helper. I will always remember those sleepless nights on top of the Physics Building!

Doug Tody and Mike Fitzpatrick from the IRAF programming group have provided much technical information on and insights into IRAF.

Petra Nass from ESO deserves a special thank you for answering technical questions I have had about MIDAS, and for encouraging me when I was under pressure for meeting the deadline.

To the good folks at Research System, Inc., thank you for providing various information on IDL. Special thanks to Kathleen Falco, Chad Bamrick and Eduardo Iturrate.

Tim Cornwell from AIPS++ has shared with me his valuable experience and insightful thoughts as the project manager of AIPS++.

Pei-Lin Liang, my beloved sister in Christ, thanks for proofreading this thesis at the last minute, and for many enlightening conversations.

Finally, thanks to all my friends and family for putting up with me as I struggled towards completion. Yes, it really is done!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 IRAF or MIDAS?

IRAF (Image Reduction and Analysis Facility) [32] and MIDAS (Munich Image Data Analysis System) [15] are the two major open source astronomical data reduction and analysis software packages on the current market. IRAF is developed by NOAO (National Optical Astronomy Observatories), with an estimated 5000 users at 1500 sites around the world [31]. MIDAS is developed and maintained by ESO (European Southern Observatory).

At the 6th Southern Photometry Conference [1] in New Zealand, the audience expressed great interest in setting up a workshop on how to use IRAF and/or MIDAS. The pros and cons of the two software packages were also discussed briefly.

This thesis was inspired by the discussion at the Southern Photometry Conference, and by my experiences using IRAF to reduce CCD images of a recently discovered supernova remnant under the supervision of Professor Phil Yock.

My goal is carry out a comparative study of IRAF and MIDAS, so that prospective users can decide which software best suits their particular needs.

1.2 An Ideal Software Package For Astronomy

1.2.1 Introduction

In the search for an ideal software package for observational astronomers, I came across the NASA Astrophysics Data System (ADS) Abstract Service [5]. The database contains over one million astronomy-related abstracts. It turns out that there is only one published paper on the topic of comparative study of astronomical software packages. I will briefly discuss this paper in the next section, in order to justify the need for another comparative study on astronomical software packages.

1.2.2 Literature Review

In 1998, Mario Pérez of Applied Research Corporation, compared various open source and commercial astronomical software packages such as IRAF, MIDAS, AIPS and IDL. His methodology was to compare each actual system with an ideal software package which has the following features:

- Simple and object-oriented philosophy
- Interpreted language (as opposed to the compile-link-test-crash-debug systems)
- Portable architecture across multiple platforms
- Adaptable to changing environments
- Robust applications

- High performance for all applications, especially graphical ones (optimal dynamic memory management) [48]

Pérez concluded that IDL was closer to the ideal than other competing products.

Briefly, Interactive Data Language (IDL) is a proprietary software system distributed by Research Systems, Inc. [53] IDL grew out of applications programs written for analysis of data from NASA missions such as the International Ultraviolet Explorer (IUE). It is therefore oriented toward use by scientists and engineers in the analysis of one, two, or three-dimensional data sets.

1.2.3 Cost Factor

If IDL is the ideal software package for astronomy, why is it that software packages like IRAF, MIDAS and AIPS++ are more widely in use by astronomers and institutions than IDL? The main concern I believe is the cost of "buy-in", that is, the effort required to adopt and to use a particular software package [40].

Financial Cost

The lowest price for an IDL educational license for a university-based researcher is USD $1647. In addition, IDL has a annual running cost (software maintenance, technical support) of USD $660. The prices of IRAF, MIDAS and IDL are summarized in Table 1.1. From this table we can see that IDL is not the cheapest option for many of the often-budget-constrained research projects in many universities. New Zealand's astronomy and astrophysics research budget per head of population is about one fifteenth of the OECD averages [11]. The United States government's FY2000 Budget Proposal would cut its Space Grant program's budget by almost one-third down to its FY98 level [21]. Such tight budgets may force researchers to search for cheaper alternatives.

Even a big institution like USA's NRAO (National Radio Astronomy Observatory) does not recommend using any commercial system as their supported system for the Green Bank Telescope for the following reasons:

- Astronomers will want to use the system at home and the cost of a commercial software system may be prohibitive for most university-based observers.
- Even if the initial cost can be met, upgrades in operating systems and in the software system will increase the amount of money outside users will need to spend.
- The commercial system still needs to be tailored to the needs of observers. That is, the cost benefits from going with a commercial system may not be that great. [52]

Development Cost

In an article published in 1994, Stephen Wampler, a software engineer for the Gemini 8-meter Telescopes Project states:

> Given the large cost of developing and subsequently maintaining in-house software, it makes sense to look for available tools. If the source code for these tools is available, then they may be useful even if they don't completely cover the required functionality.
>
> In general, adapting a generic solution to a specific problem is less costly than developing a specific solution from the ground up. Part of the reason for this is that most projects contain a great deal of software that duplicates functionality found in other projects (the "infrastructure"). A general solution typically provides the majority of this infrastructure. [67]

As mentioned earlier, IDL is more of a computer programming language with extensive scientific libraries than a ready-to-use off-the-shelf data reduction and analysis system. The users are expected to put effort into developing

Product	Platform	Software	Shipping	Software Maintenance per year	Technical Support per year	Total Cost for first year
IRAF						
CD-ROM	Unix	$40	$10	free	free	$50
ftp	Unix	free	free	free	free	free
MIDAS						
CD-ROM	Unix	$20	free	free	free	$20
ftp	Unix	free	free	free	free	free
IDL						
Single	WIN95/98 NT/MAC	$2,275	$77	$300	$360	$3,012
Education	WIN95/98 NT/MAC	$910	$77	$300	$360	$1,647
Single	Unix	$4,195	$77	$420	$360	$5,052
Education	Unix	$1,678	$77	$420	$360	$2,535
Lab. license ($\leq$15 users)	—	—	—	$1,995 license renewal		$6,545
Dept. license ($\leq$25 users)	—	—	—	$4,995 license renewal		$9,790

Table 1.1: Cost comparison summary (price in US dollars), in January 2000 [7].

their own software with IDL in order to work with their data. Therefore the software is not suitable for those researchers who just want to reduce and analyze their data.

Sharing Cost

An interesting example of adopting a proprietary system like IDL is ISO's ISAP (ISO Spectral Analysis Package) project. In a journal article published in 1997, E. Sturm from the MPE (Max Planck Institute for Extraterrestrial Physics) states:

> The data processing and data analysis software developed for ISO, the Infrared Space Observatory of the European Space Agency (ESA), is perfectly suited to demonstrate, how scientific work can benefit from the strengths and advantages of IDL. [55]

But in the same year, at the ADASS VI Conference's Interactive Data Analysis Environment BOF Session, ISO was criticized in choosing IDL:

> The point was made that if a large project chooses a proprietary system for its basic calibration environment, it makes access to their project much more difficult and/or costly to data customers who do not have licenses for that particular product. Likewise, the project places its investment in coding at risk should the provider of that product cease to support it or cease to exist. ISO was cited as having committed this error in choosing IDL. [26]

The discussion on the topic of the cost factor in this section not only contradicts Pérez's suggestion that IDL is an ideal software package for astronomy, it also poses the question: "What are the criteria for an ideal software package for astronomy?" It is the goal of this thesis to provide a measurable matrix to help readers of this thesis to decide for themselves.

1.3 Methodology and Scope

While this thesis is primarily a comparison between two software packages, IRAF and MIDAS, it also covers many aspects of computing, from software architecture design, human-computer interaction, object-oriented design, software evolution to system performance analysis.

The sources consulted include the NASA Astrophysics Data System (ADS) [5], web search results, orthodox textual sources and my interviews with various people from both astronomy and computer science communities.

These sources are used to develop a profile of current software development of astronomy. This involves a literature review on what astronomers think an ideal software packages for astronomy should be. It is followed by an in-depth comparative study of the design and features of IRAF and MIDAS. This is done by comparing documented design and features, and my personal experience with these two software packages. The future of the two astronomical software packages is discussed in Chapter 8 based on my interview with members of the two astronomical programming groups. In Chapter 9, I have provided a brief discussion on the topic of alternative astronomical software packages. My summary conclusions appear in Chapter 10.

1.4 Objective

The objective of this thesis is to carry out a comparative study of two of the most popular software packages for professional astronomers from the perspectives of a software developer and a first-time end-user. My goal is to provide the reader of this thesis with an understanding of the underlying software architecture, programming environment, and how the program performs in real life situations. Having read the thesis, prospective users can decide for themselves whether IRAF or MIDAS suits their needs, and if not, what are the alternatives.

Chapter 2

Observational Astronomy

2.1 The CCD Revolution

The world of observational astronomy and space science has undergone a revolutionary change since the invention of the Charged-Coupled Device (CCD) in 1970 at Bell Labs by W. Boyle and G. Smith. CCD is a semiconductor detector utilizing the photoelectric effect. Photons displace electrons from their atoms creating an electron hole pair. A potential difference across the pixel forms a potential well for the electrons to fall into. After the exposure, the charge in the pixel is read as a current and converted into digital units. The CCD does not detect all incident photons. The fraction of detections is its "quantum efficiency" [56].

The first imaging CCD (100x100 pixels) was produced by Fairchild Electronics in 1974. With an 8-inch telescope this CCD produced the first ever astronomical CCD image. The quantum efficiency of this device was less than 0.5%, slightly less than good photographic plates at the time. Needless to say, observational astronomers at this time were less than enthusiastic

about replacing glass plates with digital electronics. However, this perception changed in 1979 when the RCA 320x512 LN cooled CCD system saw first light on a 0.9-meter telescope at Kitt Peak National Observatory [41].

Observations made with this CCD quickly showed its superiority over photographic plates. The quantum efficiency was at least 50 times higher. The device itself was highly linear and therefore much easier to calibrate than the nonlinear photographic plates technology it replaced. Over the last 20 years, the CCD has been steadily improved and is evolving toward an ideal detector that has 100% quantum efficiency, perfectly uniform response, noiseless, unlimited dynamic range, and an unlimited number of pixels. Today's high-end CCDs are very close to being this ideal detector [8].

With these highly efficient, highly linear, large dynamic range, fairly uniform response, relatively low noise digital detectors, astronomers today are able to make cosmic discoveries that were unthinkable prior to the invention of CCDs.

2.2 Astronomical Data Reduction & Analysis

Astronomical data generally consist of either two-dimensional images (or 3-dimensional image cubes containing many images), or one-dimensional spectra. Spectra may either be temporal (light curves) or wavelength series. The basic astronomical measurements are position (astrometry) and flux (spectroscopy or photometry). Flux may be measured as a function of position, wavelength or time.

Reduction is the process of turning raw data into a calibrated product. All astronomical measurements take place against some background. This background may consist of a number of components. There may be instrumental background (e.g., read noise in a CCD) or sky background (the sky is not black), and the background must be subtracted from the measurement. For example, if you measure the flux in an emission line, it is often superimposed on a continuum. The image of an object is superimposed on a non-zero background. Since these backgrounds contribute to the signal measured by the CCD, they have an associated measurement uncertainty. These errors

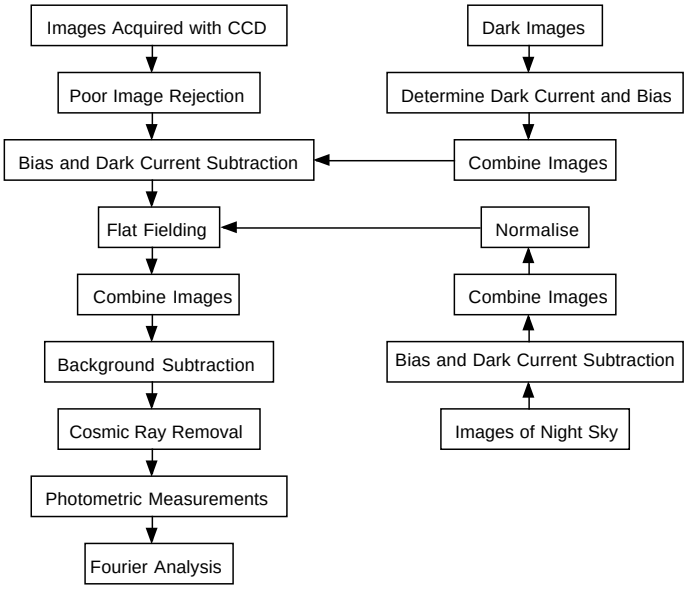

Figure 2.1: CCD Image Reduction Procedure [47]

must be removed from the images before they can be processed [47]. See Figure 2.1 for a typical process of image reduction in astronomical signal processing.

Following the reduction process is data analysis. A few of the many astronomical data analysis methods in common use are summarized in Figure 2.2. Depending on the type of data, different analytic methods are used. The three major analytic methods for optical astronomy are:

Astrometry - Astrometry measures the precise positions and motions of the Sun, Moon, planets, asteroids and stars. Observations are made to determine the fundamental celestial reference frame. Such observations provide the basis for the world standard of the star positions.

Photometry - Photometry measures the brightness of stars and other celestial objects (nebulae, galaxies, planets, etc.). It provides information on structure, temperature, distance, age of the object. Examples of photometry include (among many others) surface photometry, which in turn encompasses extended surface photometry.

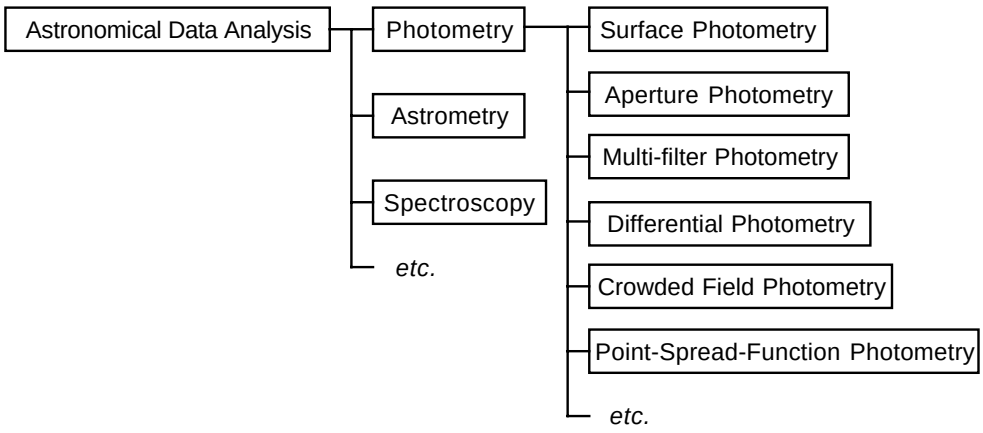

Figure 2.2: Selected Astronomical Data Analysis Methods.

Spectroscopy - The study of spectral lines from different atoms and molecules. Spectroscopy is an important part of studying the chemistry that goes on in stars and in interstellar clouds. It provides information on the radial motions, chemical compositions, and physical conditions of planets, comets, stars, and nebulae.

My impression is that there are many different analytic methods for astronomy, and each analytic method has many different variants of itself to suit the particular requirements of a researcher and instrument.

2.3 Astronomical Software Packages

So how do we reduce and analyze our data? Several well-developed software packages are available to researchers. In this thesis our study is focused on the two open source software packages IRAF and MIDAS which are primarily used for optical astronomy.

2.3.1 NOAO IRAF

The Image Reduction and Analysis Facility (IRAF) is a large software package for astronomical data reduction and analysis. It was originally developed at NOAO (National Optical Astronomy Observatories) for ground-based optical astronomy. Since then, major subpackages have been added for the Hubble Space Telescope, ultraviolet and X-ray astronomy.

Brief History

The IRAF project started in 1981 at the Kitt Peak National Observatory, two years after the RCA CCD system was used on the 0.9-meter telescope at Kitt Peak. The importance of the RCA CCD system to astronomy is discussed in Section 2.1.

In 1983, the NOAO IRAF programming group was formed, and IRAF was chosen by the Space Telescope Science Institute (STScI) to host their Science Data Analysis System (SDAS). The first internal release of IRAF occurred in 1984, and the first public release of both the Unix and VMS versions of IRAF occurred in 1986 [57].

The original system was running on VMS and BSD 4.2. It was later ported to various Unix systems, including PC-IRAF, a port to Linux/FreeBSD in 1998 [20].

Description

IRAF is widely used by the professional astronomers and observatories around the world, with an estimated 5000 users at 1500 sites around the world [31].

One reason for IRAF's popularity is the number of packages available in IRAF. It comes with packages for general image processing, graphics, and optical/infrared data reduction and analysis. It also has external or layered packages for applications such as data acquisition or handling data from other observatories. Many of these external packages are not developed by NOAO,

but by other institutions and projects like STScI (Space Telescope Science Institute), EUVE (Extreme UltraViolet Explorer) and ROSAT (ROentgen SATellite).

2.3.2 ESO-MIDAS

MIDAS (Munich Image Data Analysis System) is the European answer to IRAF. It is a portable image processing system developed at ESO (European Southern Observatory) and is used for off-line data reduction at ESO and many astronomical institutes all over Europe.

Brief History

With the introduction of CCD imaging systems to the astronomy community in 1974, it was natural for ESO to adopt the latest technology at the La Silla Observatory, and a digital data reduction software package was required. The development of ESO's first Image Handling And Processing system (IHAP) was started in 1975 by F. Middelburg. The software was based on HP 1000 computers with the RTE operating system that was used for data acquisition at telescopes.

The next generation of ESO's image processing system Munich Image Data Analysis System (MIDAS) started in 1981 by Klaus Banse, and was based on the experience from IHAP. The major goal of MIDAS is:

> To provide the astronomical community with software tools for both general image processing and analysis, and for the reduction of data acquired at ESO telescopes. [13]

MIDAS borrowed some ideas from the UK STARLINK project, and it was developed on DEC/VMS system using VAX-FORTRAN with many VAX/VMS specific features. MIDAS was in general usage at ESO and was released to other institutes in 1985 [23].

With the introduction of workstations and the acceptance of Unix operating system in the 1980s, a largely redesigned version of MIDAS was developed in 1986. It was released to the public in 1988 with the support of both VMS and Unix systems [23].

Description

The designers of MIDAS included some provisions to ensure their system can evolve in the future [17]:

Modular Design, for adapting to different astronomical instruments.

Portability, for running and migration on different computer platforms.

Use of Standards, for ensuring the future of the system, by adopting popular standards like the FORTRAN, C programming language, and X Window system.

Easy to Program, for encouraging developers to develop applications for MIDAS, by providing simple interface routines and implementing a flexible control language.

Open Design, for integrating contributed software from other institutes into MIDAS.

In a brief report on the United Nations/European Space Agency on Space Research Workshop, Professor H. Haubold noted that MIDAS is:

> built along lines which allow easy integration of complex analysis algorithms as well as allows great flexibility in interactive use and in the creation of user specific procedures from the available basic building blocks. [27]

The participants of this workshop evidently believe that the designers of MIDAS have met their goals of modular design, easy to program and open design.

In the following chapters, I will discuss various aspects of IRAF and MIDAS: software architecture, application programming environment, graphical user interface, system requirements, system installation, current status and future plans. I will also describe a comparative experiment with IRAF and MIDAS in Chapter 7.

Chapter 3

Software Architecture

3.1 Introduction

In "Software Engineering - A Practitioner's Approach", R. Pressman states:

> A proper architectural design develops a modular program structure and represents the control relationships between modules. In addition, it melds program structure and data structure, defining interfaces that enable data to flow throughout the program. [51]

In order to discuss the software components in IRAF and MIDAS, we need a common standard terminology. I have devised a standard terminology for the four major common components in IRAF and MIDAS that we will discuss in detail in the following sections. These components are:

Applications - The Applications provide the functionality to the system.

User Interface - The user interacts with the Applications via the User Interface.

Mid-level Interface - The Mid-level Interface provides a platform independent interface to the Applications, so the Applications are portable across different platforms.

Low-level Interface - The Low-level Interface deals with the complexity of the underlying host operating system.

My standard terminology and its corresponding terms for IRAF and MIDAS are summarized in Table 3.1. According to the chief programmer and designer of IRAF, Doug Tody, its four major components are the Command Language (CL), Application Programs, IRAF Virtual Operating System (VOS) and the Host System Interface (HSI) [57]. At the highest level, MIDAS has a similar architecture with its three major components: the monitor, the applications and the interfaces. The MIDAS monitor corresponds to the IRAF CL. The MIDAS interfaces consist of several general interfaces correspond to the IRAF's VOS, and the OS-interfaces to IRAF's HSI [17].

I have included a diagram of the layered structure of MIDAS from the MIDAS documentation [23] in Figure 3.2. The *SC and TC interfaces* in the diagram are the Standard Interfaces and Table Interfaces in C, and the *F-to-C interfaces* is the FORTRAN to C Interfaces. I have also created a corresponding system structure diagram for IRAF in Figure 3.1. Mike Fitzpatrick, a programmer in the IRAF staff, has kindly checked and confirmed Figure 3.1 for me [20].

Terminology	**IRAF**	**MIDAS**
Applications	Application Programs	Applications
User Interface	Command Language	Monitor
Mid-level Interface	Virtual Operating System (VOS)	General Interfaces
Low-level Interface	Host System Interface (HSI)	OS-Interfaces

Table 3.1: Standard Terminology for Software Components

In the following sections, we will study and discuss the similarities and differences of these two software packages in detail.

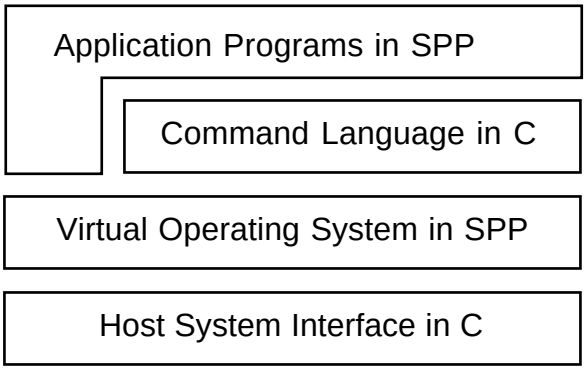

Figure 3.1: Layered structure of IRAF

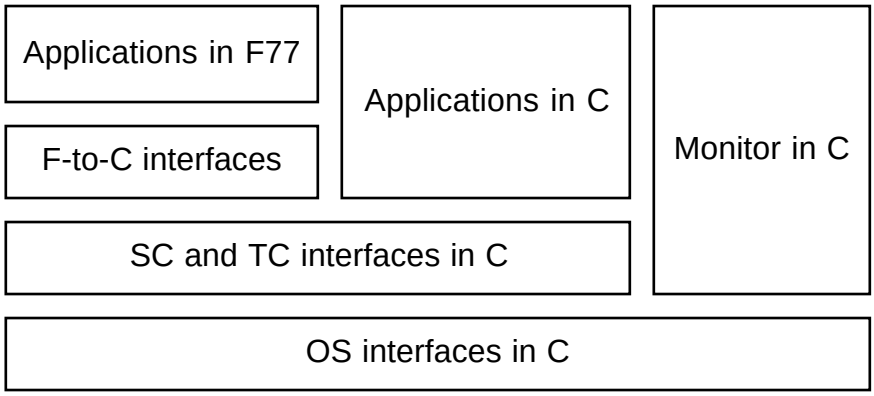

Figure 3.2: Layered structure of MIDAS [23]

3.2 Applications

3.2.1 IRAF

The IRAF application packages are divided into two classes, the system packages and the scientific reduction and analysis packages. A short summary of the system packages is in Table 3.2. Here is my summary of the system packages from IRAF's online help:

dataio - It contains file conversion utilities to import files in other format into IRAF, and export IRAF files to other file format.

images - Utilities related to image coordinates, image filtering, image fitting, geometric transformation, image matching and display are in this package.

language - This package contains all the CL language components, built-in commands and functions.

lists - A mixture of list related utilities: calculate mean, standard deviation of a list, linear transformation and read cursor position and make it a list etc.

plot - General plotting utilities, with vector plotting, surface plotting to different devices.

softools - It contains utilities to make help pages, manual, make or compile packages, read/write TAR archive files etc.

system - Basic file/directory related operations (create, delete, display etc.), printing utilities, and system environment utilities are included in this package.

utilities - Miscellaneous utilities such as random number generator, convert a file to upper/lower case and data fitting etc.

There are also two categories in the scientific reduction and analysis packages. The first category belongs to the packages that are intended for general use. Since these general packages are instrument independent, they are not the most convenient packages to use for reducing data for a particular instru-

ment. Therefore, the second category includes the packages that are designed for specific instruments. Usually these are just CL scripts that fetch instrument specific parameters from the image headers and call tasks in the general packages.

The standard IRAF distribution includes the core IRAF and NOAO package trees. It contains about 50 packages, or several hundred tasks. In addition, many third party packages have been developed by other institutions and groups. Samples of scientific reduction and analysis packages are included in Table 3.3, again, I have made a summary for each package from the IRAF online help:

artdata - It includes utilities to create artificial stars/galaxies lists, artificial noise and cosmic rays, artificial 1D/2D spectra etc.

digiphot - Utilities to perform aperture photometry, crowded field photometry, photometric calibration and some miscellaneous photometry tools.

imred - This package includes general CCD image reduction utilities and NOAO instruments reduction utilities.

onedspec - Utilities for 1D spectra reduction and analysis.

twodspec - Aperture extraction, longslit utilities for 2D reduction and analysis.

Description	**Package name**
Data input and output	dataio
General image processing, display	images
The command language itself	language
List processing	lists
General graphics utilities	plot
Software tools, system maintenance	softools
System utilities (file operations, etc.)	system
Miscellaneous utilities	utilities

Table 3.2: IRAF System Packages

3.2.2 MIDAS

The MIDAS applications are also divided into two classes, the general system functions and special purpose functions. General system functions are always defined in the user environment, whereas the special purpose functions can be enabled or disabled on an individual basis.

I was unable to obtain specific package details for the MIDAS general system functions, but I have found a summary of the general functions in the MIDAS documentation [39]:

Image display package - It includes functions to display image, zoom and scroll, getting cursor values, blinking, overlaying graphics, working in PseudoColour or RGB mode, printing images etc.

Plotting package - It has device independent functions to present data in a graphical format, and for interactive data reduction.

Data I/O - Data transfer in IHAP and FITS format.

General image processing - It contains functions such as filtering, resampling, interpolation, rotation, extraction/insertion and other general image processing.

Table system - It has functions which process tabular data.

Fitting package - It offers tools to fit nonlinear functions and model data distributions.

Spectral analysis - It is an instrument independent spectral data reduction and analysis.

Description	Package name
Artificial data generation package	artdata
Digital photometry package	digiphot
NOAO Instrument reduction package	imred
1D spectral reduction and analysis package	onedspec
2D spectral reduction and analysis package	twodspec

Table 3.3: Samples of IRAF Reduction and Analysis Packages

The special purpose functions are astronomical specific applications that are available as individual packages inside the system. Those are called the MIDAS contexts. There are three special features to these contexts [39]:

- The software assumes very little about the actual instruments. Instrument dependent characteristics in the data are put into a normalized form that can evolve towards an instrumental database.
- Different modules communicate through intermediate variables and tables that are based on the relational database model. This allows exchange of modules between contexts and to reuse the algorithms in different reduction schemes.
- User's session can be defined by set of keywords and associated calibration information.

The standard MIDAS distribution includes about 35 packages. Some third party external packages have also been developed. Samples of the MIDAS contexts are included in Table 3.4, and a brief introduction to each context is given below [39]:

geotest - It generates geometric test patterns to test algorithms that do not maintain the identity of a pixel, such as rebinning, rotation and filtering.

astrometry - This context includes several commands to make different standard coordinate transformations and projection of positional data into different systems. It is also used for astrometry measurements.

photom - It includes several photometric reduction utilities.

statistics - It offers functions such as multivariate analysis, clustering techniques and discriminant analysis methods.

cloud - It models absorption lines produced by interstellar or intergalactic clouds.

real - This context provides relational algebra operators on tables.

tsa - The Time Series Analysis context is used for period determination.

model - This context comes with two large databases: technical data describing the HST instruments, and an astrophysical data catalogue. It is possible to simulate HST observations with different instruments with this context.

ccd - It contains functions for doing relative pixels calibration, averaging images and removing instrumental faults.

irac - It is used for the analysis of image data observed with the ESO infrared camera.

Description	Package name
Generation of test images	geotest
Astrometry package	astrometry
Photometry package	photom
Multivariate methods for statistical analysis	statistics
Model interstellar or intergalactic absorption clouds	cloud
Relational algebra on tables	real
Tools to do a time series analysis on data	tsa
Simulation of HST observations with different instruments	model
Calibration and reduction of 2D photometric CCD images	ccd
Reduction of images observed with the infrared camera	irac

Table 3.4: Samples of MIDAS Contexts

3.2.3 Comparison

Figure 3.3 shows how the applications are organized in IRAF and MIDAS. The basic structure is similar, with system related applications and astronomical specific applications. I noted the differences in the way instrumental dependency are implemented by the two software packages, and the number of applications developed for each package. My comparison is summarized in Table 3.5, and is discussed below:

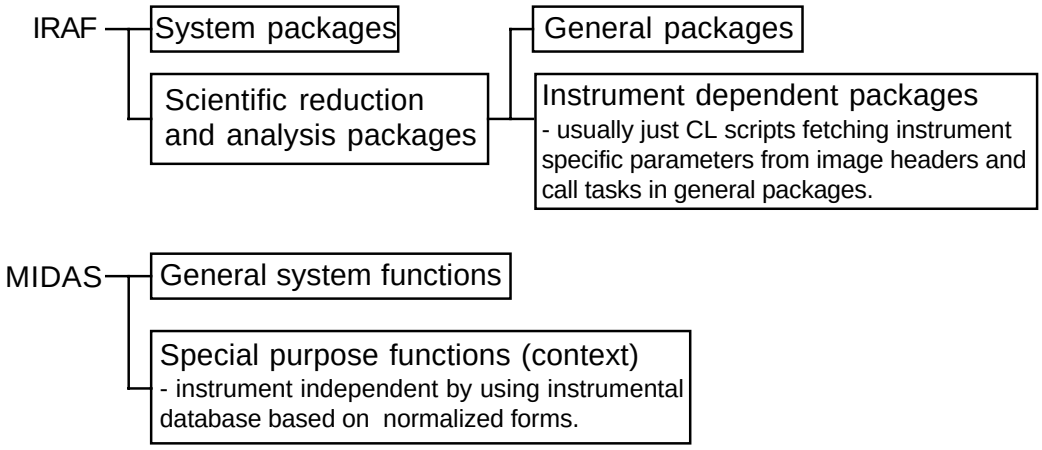

Figure 3.3: IRAF and MIDAS Applications Summary

	IRAF		MIDAS	
	Comment	Score	Comment	Score
Instrumental Dependency	CL script and specialized package	0	Normalized form/ Instrumental database	1
Number of Applications	More	1	Less	0

Table 3.5: Applications Comparison Summary

Instrumental Dependency - The IRAF and MIDAS group use different tactics to solve the instrumental dependency problem. The IRAF group solves this problem by developing a specialized package for the particular instrument, or using a CL script to read instrument specific parameters from image headers and calling tasks in the general packages.

On the other hand, MIDAS uses a normalized form to store instrument dependent characteristics. An instrumental database can be constructed from a collection of the normalized forms. This, in my opinion, is a more elegant solution, as the instrumental database can be of general use.

Number of Applications - In my search of the literature, I found no direct evidence of whether more applications have been developed for IRAF or for MIDAS. My best indication is that IRAF has the larger user community, because the ADS (Astrophysics Data System) abstract database [2] shows there are 78 papers with the word 'IRAF' in their titles, and 32 papers for 'MIDAS' at the time this thesis is being written.

It is beyond the scope of this thesis to compare the individual reduction and analysis packages provided by MIDAS and IRAF.

3.3 User Interface

3.3.1 Command Language

IRAF

The IRAF Command Language (CL) is designed to serve both as a command language and as an interpreted programming language. The CL uses C style expressions and control constructs. Its basic command syntax is the same as that used in the Unix *csh*. Other features of CL that are similar to *csh* are i/o redirection, background job, history and pipe facilities [57].

A distinctive feature of IRAF CL is that it allows flexible syntax for both interactive environments and scripting. An example of this is seen in function calls. A function call in a typical non-interactive script has the form `function (arg1, arg2 , ...)`. But in interactive environments (such as shells), spaces are used as delimiters instead of parentheses and commas [26]. IRAF CL allows both styles.

MIDAS

The MIDAS Command Language (MCL) is sometimes called the MIDAS Control Language in the MIDAS documentation. It serves as a command language and as an interpreted programming language. It is modeled after the VAX/VMS Digital Command Language (DCL) for historical reasons, as the original MIDAS is developed on VAX/VMS. It also provides background job, history and pipe facilities [39].

Programs written in MCL are called procedures. In fact, except for the system commands, all MIDAS commands are MCL procedures themselves. An important feature of MIDAS is a debugger to debug MCL procedures as well as application code in FORTRAN-77 and C. Basic debugging commands like setting/clearing break points, listing preprocessed code, stepping, inspecting keywords (variables) are provided. It helps developers with writing long and complicated procedures.

The syntax of a MCL command is as follows:

```
COMMAND/qualifier parameter
```

where:

- `COMMAND` is the MCL command verb.
- `qualifier` is always preceded by a slash character (/). The qualifier modifies the operation of the command. Not all commands have qualifiers; some have more than one qualifier. Many commands have default qualifiers.
- `parameter` is the object upon which the command acts. Not all commands need a parameter; some require more than one parameter.

Here is an example of MCL command to load an image of the Sombrero galaxy:

```
Midas 001> LOAD/IMAG /midas/data/demo/sombrero.bdf
```

Comparison

There are two distinctive differences with the two command languages: the syntax and availability of debugging tools. My summary of the comparisons on the CL Syntax and debugger can be found in Table 3.6.

CL Syntax - Both IRAF CL and MCL were designed in the early 1980s, and had adopted the syntax of popular contemporary command languages like *csh* by Bill Joy at UC Berkeley, and VAX/VMS Digital Command Language (DCL). My opinion is that due to the popularity of the C programming language, new users will find IRAF CL's C style syntax easier to pick up than MIDAS' DCL style syntax.

Debugger - One advantage of MCL over IRAF CL is the availability of the debugger. The debugger allows easy testing for long and complicated CL scripts. Unless the script is trivial, a debugger is an essential tool for software development.

Future Trend for CL

Both IRAF CL and MCL were designed specifically for astronomical environments, and not as general programming languages. This limits their use in an interactive environment. There is a movement in the astronomical community for astronomical software systems to adopt the syntax and the features found in more modern general purpose scripting languages (including Glish [54], Perl [34], and Python [49]). Examples of this trend are:

- The AIPS++ project is using Glish as the basic command-line interface and as the backbone for internal control and communication. A *Tk* binding for Glish has also been developed to provide a GUI for AIPS++. Glish is discussed in Section 9.3.4.
- The Space Telescope Science Institute (STScI) is currently working on a Python version of IRAF CL [20].

	IRAF		MIDAS	
	Comment	Score	Comment	Score
Syntax	csh	1	VAX/VMS DCL	0
Debugger	n/a	0	Available	1

Table 3.6: Command Language Comparison Summary

3.3.2 Shell

IRAF

The IRAF Command Language (CL) also serves as the user's interface to the IRAF system. Some of the IRAF CL features are:

- provides a uniform environment on all host systems
- package structure for organization and extensibility
- menus and extensive online help facilities
- concise command syntax similar to Unix *cshell*
- i/o redirection and pipes; aggregate commands
- minimum match abbreviations for task and parameter names
- both local and global parameters, hidden parameters
- direct access to host system; foreign task interface
- parameter set editor; command history editor
- background job submission (including queuing)
- logfile facility for recording all task invocations
- graphics and image display cursor mode facilities
- virtual filename facility; Unix style pathnames to files
- programmable procedures, C styles expressions and control constructs [57]

Some of the above IRAF features are discussed in Chapter 7, in which IRAF will be used to reduce some real datasets. An example of IRAF's minimum matching mechanism is also shown at the end of the next section.

MIDAS

The user's interface to MIDAS is called the monitor. All the user interaction and the scheduling of processes to execute the commands are done through it. The monitor provides the following functions:

- display online help with different levels of detail
- keep a log of all operations during a session
- serve as a command interpreter
- preprocess input strings to translate user-defined symbols and to facilitate command abbreviation (minimum matching), command numbering (enabling easy command repetition and editing), command buffering, shorthand last line usage and multiple commands per line
- execute applications in a subprocess [39]

An important feature of MIDAS is the command abbreviation. It is very similar to the command and file name completion feature in Emacs or *tcsh* shell. The following examples demonstrate the way IRAF and MIDAS implement their command abbreviation. We are going to load and display an image of the Sombrero Galaxy.

The first example shows how to use the command *display* in IRAF to display the image. I typed *dis*, but this resulted in an error message, since there are more than one command name that begins with *dis*, so an error message was given. It should be noted that the error message does not provide more information to the user to accomplish the task. I tried again, this time with *disp*. The CL recognized that this is a unique abbreviation for the command *display*, so it executed the *display* command and asked for the name of the image file to be display.

```
cl> dis
ERROR: ambiguous task ‘dis’
cl> disp
image to be displayed: sombrero.fits
```

The second example shows how to use the command *LOAD/IMG* in MIDAS to display the image. I typed *LO* and pressed the TAB key, since there are more than one command names that begin with *LO*, a beep was given that tells me that abbreviation *LO* does not match to a unique command. I pressed the TAB key twice more, and MIDAS displayed a list of eight command names that begin with *LO*. Now I can either type the rest of the command or type the following key sequence: *A*, TAB, *IM*, TAB to complete the command name matching. It should be noted that the MIDAS command abbreviation also works for file names.

```
Midas 001> LO
LOAD/IMAG  LOAD/LUT   LOCK/KEYW  LOG/ON
LOAD/ITT   LOAD/TABL  LOG/OFF    LOG/TOF
Midas 001> LOAD/IMAG
Enter input image: sombrero.fits
```

Comparison

Both IRAF and MIDAS are command-driven systems, with many similarities and some differences. A few features that I considered to be important are command/file name completion, online help, pipes and i/o redirection. These features are discussed below, and a summary of the discussion can be found in Table 3.7.

Command and File Name Completion - One thing that stands out from MIDAS' features is the way command abbreviation is implemented. This feature is in my opinion the most important addition to the command-line interface. It helps to minimize keystrokes and command memorization. Although IRAF also has minimum match abbreviations similar to MIDAS' command abbreviation, it is not as elegant for the users as MIDAS.

Online Help The GUI integration into MIDAS means the user can launch the GUI frontend of the online help, XHelp, which is independent from the MIDAS monitor. In this way, documentation can be accessed even when the monitor is active. To achieve similar results in IRAF, the user will need to start a new IRAF session in a new window to access online help.

Adjustable Help Detail - The user can control the length and detail of the online help in MIDAS by changing the user levels (novice, user and expert). The detail can vary from single line text to many pages of extensive descriptions of application algorithms. On the other hand, IRAF's extensive online help is organized in the order of syntax, description and examples which are very similar to the Unix *man*.

Pipes & i/o Redirection A seasoned Unix user will find the Unix-styled i/o redirection and pipes in IRAF very familiar. MIDAS also has i/o redirection and pipes, but they are not implemented in the Unix style, instead they use the MIDAS keywords.

	IRAF		MIDAS	
	Comment	Score	Comment	Score
Command and File Name Completion	Minimum Matching	0	Command Abbreviation	1
Online Help	Text	0	Text/GUI	1
Adjustable Help Detail	n/a	0	User-defined	1
Pipes & i/o Redirection	Unix Style	1	Keywords	0

Table 3.7: Shell Comparison Summary

3.4 Mid-level and Low-level Interfaces

3.4.1 Introduction

At the beginning of this chapter, it is mentioned that architectural design defines the interfaces that enable data to flow throughout the program. In this section the interfaces in IRAF and MIDAS that provide the link from the systems both to the applications and to the underlying operating systems will be discussed.

3.4.2 IRAF

Mid-level Interface: VOS

The Virtual Operating System (VOS) is IRAF's programming interface to application developers. Doug Tody, the chief programmer and designer of IRAF, states:

> The primary functions of the VOS are to provide all the basic functionality required by applications programs, and to isolate applications programs from the host system. The VOS defines a complete programming environment suitable both for general programming and for scientific programming in particular. [57]

There are many subsystems in the VOS. The following is a brief summary of an explanation for the major VOS subsystems taken from IRAF documentation [57]:

CLIO - It consists of a few get/put procedures for CL parameter i/o.

ETC - It contains miscellaneous facilities like exception handling, process control, device allocation, symbol tables etc.

FIO - It is a device independent file i/o interface to both text files and binary files.

FMTIO - This interface is concerned with formatting output text and decoding input text.

GIO - It provides a uniform, device independent interface for vector graphics and image display and printing.

IMIO - It is used to access bulk data arrays or images stored in binary files.

KI - It contains kernel interface for network communications.

LIBC - Unix *stdio* emulation, C binding for the VOS, used by the CL.

MEMIO - It provides dynamic memory allocation and memory management facilities to IRAF. Both heap and stack facilities are provided.

MTIO - It contains magtape i/o interface.

OSB - It contains bit and byte primitives.

TTY - It provides a basic screen management for terminals.

VOPS - It is a library of subroutines to perform simple vector operation on one or more 1D arrays.

Low-level Interface: HSI

The Host System Interface (HSI) is the interface between IRAF and a particular host operating system. To port IRAF to a new operating system, HSI must be implemented for that particular operating system.

The following is a summary of the five components of the HSI from the IRAF documentation [57]:

IRAF kernel - It is a subroutine library containing all the host dependent primitive functions required by the VOS, and is the most machine dependent part of the HSI.

bootstrap utilities - It contains a set of utilities required to compile and maintain the main IRAF system.

hlib library - It contains a number of host and site dependent compile and runtime tables used to parameterize the characteristics of the host system.

as directory - It contains the assembly code for some library modules.

gdev directories - The directories contain the host dependent i/o interfaces for any binary graphics devices supported by GIO kernels.

3.4.3 MIDAS

Mid-level Interface: General Interfaces

The general interfaces bind the applications to the monitor, and define the possible interaction with application tasks and monitor. There are four sets of general interfaces in MIDAS [14]:

Standard Interfaces - for general i/o and image access

Table Interfaces - for access to table structures

Graphics Interfaces - for generation of graphical presentation of the MIDAS data structures.

Communication Interfaces - To allow access by other processes to the services and utilities provided by MIDAS.

The general interfaces in MIDAS have the same functions as the Virtual Operating System (VOS) in IRAF. Both provide interfaces to basic functionality required by application programs, and isolate application programs from the host system. In the case of MIDAS, only general interfaces and monitor are allowed to access the OS-interfaces, which deals directly with the host operating system.

A complete reference to the MIDAS general interfaces libraries can be found in *ESO-MIDAS Environment Document*, which is also available on the web [14].

Low-level Interface: OS-Interfaces

Very little information is available on the MIDAS OS-interfaces, but Carlos Guirao from ESO tells me:

> Unfortunately there is not a single written document that describes in detail the MIDAS OS implementation. Source files in these directories are documented to the level needed by developers of MIDAS monitor and other interfaces. [43]

According to the information I gathered, the OS-interfaces is the only system dependent layer of MIDAS. Standard versions of OS routines exist for different Unix implementations and for VAX/VMS. To port MIDAS to a new platform, only the routines in the OS-interfaces need to be rewritten. The rest of the system will remain the same.

Apart from the portability issue, another important reason for providing an OS-interfaces is to simplify application programming in MIDAS. Again, Carlos Guirao provides a very helpful explanation:

> Not only portability was the main reason for the MIDAS OS-interface, but also, and probably more important, the creation of a layer that shield the sometimes much more complex OS programming language, like access to devices and system resources. [43]

In general, the OS-interfaces in MIDAS corresponds to the Host System Interface (HSI) in IRAF.

3.4.4 Comparison

Arisen from the comparative study are two issues of interest: the interfaces architecture and the interfaces library. However it is beyond the scope of this thesis to compare the details of the individual interfaces for IRAF and MIDAS. The two issues are summarized in Table 3.8.

Interfaces Architecture - IRAF and MIDAS are similar in their interfaces architecture. The VOS of IRAF corresponds to the general interfaces of MIDAS, and HSI corresponds to the OS-interfaces. In general, porting can be done simply by implementing a new HSI or OS-interfaces for the new platform.

Interfaces Library - Since MIDAS' interfaces library is available in C and FORTRAN, this allows compiled code and interactive languages with dynamic linking capability to use those routines in the library. On the other hand, IRAF's SPP language is the only programming language that has full access to the VOS, with a small subset of VOS available to C. Therefore MIDAS is much more extendible in this regard.

	IRAF		MIDAS	
	Comment	Score	Comment	Score
Interfaces Architecture	Similar	1	Similar	1
Interfaces Library	Only SPP has full access to VOS	0	C and FORTRAN Libraries for dynamic linking	1

Table 3.8: Mid-level and Low-level Interfaces Comparison Summary

3.5 Summary

IRAF and MIDAS have very similar software architecture design, but are very different in their implementation. Both packages scored evenly in the area of Applications and Command Language. MIDAS has advantages over IRAF, because MIDAS' C/FORTRAN library allows dynamic linking, and its shell is much more user-oriented than IRAF's.

Table 3.9 is a summary of the comparisons made in this chapter. A detailed discussion on each entry is available in its respective sections in this chapter.

	IRAF		MIDAS	
	Comment	Score	Comment	Score
Applications	Fair	1	Fair	1
Command Language	Fair	1	Fair	1
Shell	Poor	0	Good	1
Mid/Low Interfaces	Fair	1	Good	1
Overall	Fair	3/4	Good	4/4

Table 3.9: Software Architecture Comparison Summary

Chapter 4

Application Programming Environment

4.1 Introduction

In his article "Scripting: Higher-Level Programming for the 21st Century", John K. Ousterhout suggests three situations in which a system programming language is preferred to a scripting language:

- The application implements complex algorithms or data structures.
- The application manipulates large datasets, for example, all the pixels in an image, such that execution speed is critical.
- The application's functions are well defined and slow to change. [35]

In astronomical computing, we use complex algorithms to reduce and analyze huge datasets taken by radio/optical telescopes. The algorithms or the analysis methods are well defined and slow to change, therefore, according to John K. Ousterhout, a system programming language should be preferred. Furthermore, simple scripts are used to control and automate sequences of commands that the astronomers frequently used. Scripting languages are available to the astronomers for this purpose.

Both IRAF and MIDAS provide different languages for scripting and application programming. The scripting language is called the Command Language in IRAF and MIDAS. A discussion on Command Languages is in Section 3.3.1. The system programming language allows the development of complex algorithms or routines. IRAF uses the SPP (Subset Pre-Processor) language, and MIDAS uses standard ANSI FORTRAN-77 and C.

4.2 IRAF Programming Environment

4.2.1 SPP

The SPP (Subset Pre-Processor) language is the native programming language for IRAF. Its designer, Doug Tody, is also responsible for the design and management of the IRAF core system. A short introduction to the SPP language is given below:

> The SPP language is based on the Ratfor language. Ratfor, in turn, is based on FORTRAN, with extensions for structured control flow, etc. The lexical form, operators, and control flow constructs are identical to those provided by Ratfor. The major differences are the data types, the form of a procedure, the addition of inline strings and character constants, the use of square brackets for arrays, and the task statement. In addition, the SPP I/O facilities provided are quite different and are tailored to the IRAF environment. The syntax of the SPP language is fairly

> straightforward and fundamentally similar to most other high-level languages. While it is based on the Ratfor language, there are elements of C as well as elements of FORTRAN. SPP is a preprocessed language. That is, there is no SPP compiler per se, but it is translated into another compilable language. In fact, SPP is first translated into Ratfor, which is processed into FORTRAN. The *xc* compiler performs all preprocessing, compilation and linkage. [38]

The IRAF group argues that since SPP is a machine and device independent programming language, code portability is achieved without "an heroic sacrifice or transcendent wisdom on the part of the programmer to produce portable code" [57]. But the IRAF group also admits openly that "SPP language and associated programming environment will never see widespread usage like C and FORTRAN." [57] Therefore, it can be concluded that the support for development in SPP is weak in comparison with C and FORTRAN. The lack of a native compiler and a source-level debugger for SPP are two good examples of the lack of programming tools available to SPP. The programmer has to use the C or FORTRAN debuggers, then translate back to the line of SPP code that is causing the problem [19].

At the moment, VOS facility and IRAF standard application libraries can only be fully accessed by programs written in SPP. Therefore, developers must learn SPP to develop their programs that exploit the full capabilities of IRAF. The other two programming choices in IRAF, with limited capabilities, are the Host FORTRAN Program Interface and the C Language Interface, which will be discussed in the following sections.

4.2.2 Host FORTRAN Interface

The Host FORTRAN program interface (IMFORT) is a small FORTRAN callable library, which consists of only a dozen or so routines. It may be linked with FORTRAN (or C) programs to get the foreign task command line from the IRAF CL, and perform some operation upon an IRAF image or images.

The IMFORT interface allows existing FORTRAN programs to be used in the IRAF environment with some modification. It is also useful for scientists who need to write a small program and do not want wish to take the time to learn how to use the SPP/VOS environment. However it is not suitable for development of large programs, as it only has very few routines and does not provide access to the IRAF VOS environment [57].

At the time of this writing, a full FORTRAN applications programming interface is not yet available to scientists/programmers who need more than the IMFORT interface but are unwilling or unable to invest the time required to learn to use the SPP environment. This problem will be addressed with the *OpenIRAF* initiative, which will provide language bindings for several programming languages (e.g., FORTRAN, C, C++, Java) [60]. However, this is a multi-year project, started in 1998, which has yet to release a binding for any language. I believe it will be at least a year before a FORTRAN language binding is available.

4.2.3 C Language Interface

The IRAF C language interface (library LIBC) consists of a Unix *stdio* emulation, and a C binding for a system programming subset of the IRAF VOS. It is used currently only to support the CL, which is written in C.

Doug Tody, the IRAF chief programmer, explains the reasons for the limited use of the C language interface:

> The C language interface could in principle be expanded to include more VOS facilities, but the sheer size of the VOS and of the rest of the programming environment makes this impractical. In any event, the SPP language is more suited to scientific programming, avoids the portability problems of calling FORTRAN library procedures from C, and will always be better integrated into the IRAF programming environment. The use of the C language interface is not recommended except possibly for porting existing large systems programs written in C to IRAF. [57]

As aforementioned, a C language binding is planned in OpenIRAF to allow the developers to use C as their programming language to develop IRAF routines and algorithms. However, there is still little known about the delivery date.

4.3 MIDAS Programming Environment

In contrast to IRAF's proprietary SPP programming environment, MIDAS applications can be developed with FORTRAN or C using the MIDAS interface libraries. There are four general interfaces in MIDAS [14]:

Standard Interfaces, for general i/o and image access

Table Interfaces, for access to table structures

Graphics Interfaces, for generation of graphical presentation of the MIDAS data structures.

Communication Interfaces, for allowing access by other processes to the services and utilities provided by MIDAS.

Section 3.4.3 contains a detailed discussion on the MIDAS interfaces. A more comprehensive references to the MIDAS interface libraries can be found in *ESO-MIDAS Environment Document* which is also available on the web [14].

To develop applications for MIDAS, the developers are required to write in standard ANSI FORTRAN-77 or C, and obey the MIDAS coding rules to achieve code portability. The MIDAS coding rules for FORTRAN and C are included in Appendix B. The advantage of developing in popular programming languages like C and FORTRAN is that there are many development tools like editors, debuggers, libraries and version control tools available to assist the developers.

4.4 Comparison

The primary difference between the IRAF and MIDAS application programming environments is the choice of programming languages. The designers of IRAF created their own specialized scientific programming language SPP. The designers of MIDAS chose to adopt general purpose programming languages like C and FORTRAN.

Three issues in a programming environment that I consider are important for a developer: programmer productivity, code efficiency and portability. A summary of the comparison made is provided in Table 4.1, and below are justifications for each entry in the table:

Programmer Productivity - As the IRAF group has admitted, SPP will never see widespread usage like C and FORTRAN, and support for its development tool is weak. On the other hand, both C and FORTRAN have many development tools available to assist the developers. From a developer's perspective, the lack of development tools is a major weakness for the IRAF application programming environment.

Code Efficiency and Optimization - The compiled SPP code could be faster than C/FORTRAN code if the translator does complex analysis of the code before translation. However, if it uses a naive translator, then we can expect the compiled code runs in the same or a slower speed than the conventional C/FORTRAN code. In most cases, the latter is true [28].

In my research, very few complaints have been made by users about IRAF's and MIDAS' speed of program execution and code efficiency. Therefore, my conclusion is that the program's speed of execution is not an apparent issue for the users of both IRAF and MIDAS.

Code Portability - The IRAF group developed their own platform independent programming language to achieve code portability. The MIDAS group's approach is to require the programmers to write their programs that conform to standard ANSI FORTRAN-77/ C specifications and coding rules set by the MIDAS group.

In my opinion, it is up to the individual programmers to decide which approach is preferred. Some may be unwilling to learn a new language unless there are overwhelming reasons to do so, others may be able to adapt to any programming language without much trouble.

	IRAF		MIDAS	
	Comment	Score	Comment	Score
Programmer Productivity	Lack of development tools	0	Many development tools	1
Code Efficiency and Optimization	No apparent problem	1	No apparent problem	1
Code Portability	Device independent programming language - SPP	1	FORTRAN-77 and C with coding rules	1
Overall	Fair	2/3	Good	3/3

Table 4.1: Programming Environment Comparison Summary

Chapter 5

Graphical User Interface

5.1 Introduction

In the late 1970s, Alan Kay and his associates at the Xerox's Palo Alto Research Center in California developed a powerful personal workstation known as the Star. Most significantly, the system was fitted with a high-resolution color display and a mouse, which allows for more sophisticated interactions than before. As a result, it became possible to point to options on a menu displayed on the screen instead of typing commands. In later versions icons on the screen were used to represent objects and functions. These could be moved around on the screen and be selected or opened by manipulating the mouse.

Unfortunately, Xerox didn't capitalize on its invention. Instead, another computer company made the most of this opportunity. In the early 1980s, Apple computer Inc. developed the Apple Lisa. Later Apple developed a smaller, cheaper and more powerful computer, the Macintosh. The success of Macintosh was phenomenal. In the late 1980s most of the operating systems

were based on the Star - Macintosh style of interface. Nowadays, every major operating system comes with some sort of Graphical User Interface (GUI) [50].

There are many advantages of GUI over the traditional command-line interface, which make them attractive even for experienced users [6]:

- Better visibility and control over specialized applications.
- Minimized syntax problems and risks of mistyping.
- Easier access to the online help.

Examples which illustrate the above points are: IRAF which has *ximtool*, a GUI interface that gives the user better visibility and control over its image display server. MIDAS has a GUI frontend to the online help to provide easy access to its help material.

There are different approaches toward GUI development. In MIDAS, standards like the OSF/MOTIF and various public-domain/commercial GUI builders (e.g., *UIDS*, *Dirt*, *Serpent*) are used to develop their GUI. In IRAF, we find a special purpose widget server based on the *Tcl* interpreter and *Xt*-based widgets. In the following sections, we will discuss each approach in detail.

5.2 IRAF Graphical User Interface

5.2.1 Introduction

In an article published in 1993, the designer and chief programmer of IRAF, Doug Tody admitted that:

> In general, the IRAF system circa 1992 is very strong in terms of the functionality provided, but is weak in the area of user interfaces. [58]

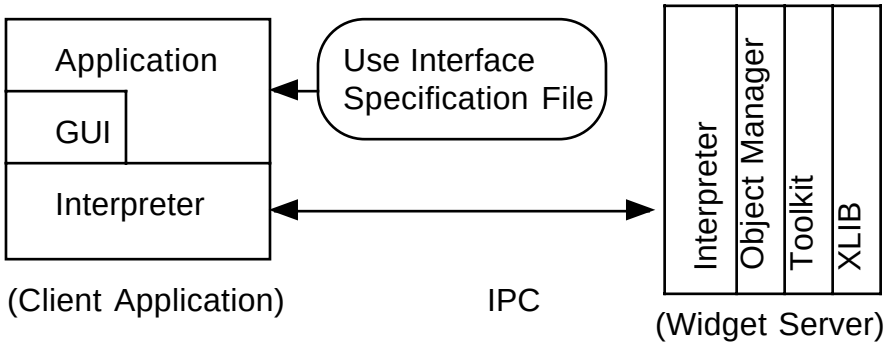

Figure 5.1: IRAF Widget Server Architecture

Even until today, this statement still holds true when one compares IRAF to other major astronomical software packages (IDL, MIDAS, AIPS++). I will discuss the IRAF group's approach to GUI implementation in the following sections.

5.2.2 The IRAF Widget Server

A short description on the origin of IRAF Widget Server is given in the published paper, "IRAF in the Nineties" by Doug Tody, the chief programmer of IRAF:

> After considerable time spent studying window systems and graphics user interfaces we think we have found a solution to this problem. It is called the widget server. In the widget server architecture the application and the user interface are in two separate processes. The application is a type of minilanguage with a simple parsed command line interface. The user interface resides in the widget server process. When an application starts up it downloads a text file to the widget server containing the user interface

> to be executed. This defines all the widgets forming the user interface as well as the code to be executed (interpreted) while the user interface executes. During execution, the user interface (widget server) and client application exchange commands and data via interprocess communication. [58]

During the execution, the client application waits for and executes requests from the GUI, and sends messages to the GUI to inform any *client events*, or changes in the state of the client. As far as the client is concerned, the GUI is a mere block of text (approximately 10Kb) to be passed on to the widget server. The GUI code is isolated from the client application. The client application only knows the messages and requests used to communicate with the GUI.

While to the user the GUI appears to be an integral part of the application, the actual compiled client application has an interpreted command line interface and can be executed stand-alone without any GUI [59].

The widget server architecture also provides window system and toolkit independence, since only it knows a particular window system or toolkit. Consequently, it is just a matter of implementing a new widget server to support a new window system or toolkit. An example is that the current IRAF widget server uses *Xt* and *Athena* widget along with their custom graphics and imaging widget, but the IRAF group is planning to switch to the *Tk* widget set for the next major release. The GUI files will remain the same, just a new widget server that supports *Tk* widget will be implemented [20].

Several software applications were developed using the widget server, and were released in a single distribution called *x11iraf*. Even though it provides the Graphical User Interface to IRAF, it is not part of the IRAF distribution and has to be downloaded separately:

> Despite the name of the software, it [*x11iraf*] is not tied to IRAF in any way, other than that it is a product of the IRAF project and is used for IRAF GUIs. [59]

The most frequently used software in *x11iraf* is *ximtool*, which is an image display server used by IRAF to display and interact with images. Nonetheless, there is a serious design flaw. *ximtool* can only use an 8-bit PseudoColour display. The IRAF FAQ explains:

> When *ximtool* was first written, about 5-6 years ago, 24-bit displays were still only available on high end systems or specialized platforms such as SGI, most PCs in the world also lacked the video cards for 24-bit and Linux was still in its early stages. Nowadays it's hard to buy a new system that doesn't have 24-bit support. It was always thought that 24-bit support would be added but it wasn't deemed a priority at the beginning when the *Gterm* widget was being written.
>
> The problem is not that it's difficult to display an image on a 24-bit screen, it's that unlike programs such as *XV* (which display a static image), *ximtool* has to allow brightness/contrast stretching when viewing the image. This is currently done, for speed reasons, by recomputing and rewriting the colormap used to display the image (rather than redisplaying the image itself). However, in a 24-bit display there is no colormap so modifying the image to increase the contrast becomes non-trivial and more expensive. There are plans to make *ximtool* and the *Gterm* widget on which it relies work under a 24-bit display, this will be part of the *x11iraf* work in progress now, but I cannot give you a date for when this will be ready. [31]

Therefore, in order to use *ximtool* to display and manipulate with IRAF in any modern workstations or PCs, the only choice is to configure an 8-bit display. This is done by starting the X session in PseudoColour mode, and switching back to 24-bit when *ximtool* is not needed. For any user who requires a 24-bit display, this is a major inconvenience. The lack of commitment by the IRAF group to fix this problem is also discouraging.

5.2.3 GUI Development

In a discussion with Paul Warhurst, I have realized that IRAF lacks a very important feature of MIDAS. The feature is a GUI which helps the user to create a mosaic of two images. I started to look into the possibility of developing a similar GUI for IRAF. I asked Doug Tody, the chief programmer of the IRAF group for his opinion on the time estimated for a seasoned programmer to develop such GUI for IRAF. Specifically the time required to learn SPP, IRAF widget server, Tcl and develop the interactive GUI task. He replied:

> It is very hard to say. A lot would depend upon the attitude of the programmer and whether they really wanted to get new task working. Maybe several weeks. A lot also depends upon the complexity of the task, e.g., what you describe could be a several person-month project in its own right depending upon how ambitious the project is. [62]

5.3 MIDAS Graphical User Interface

5.3.1 Introduction

One major goal in MIDAS' design is the use of standards. In the case of display and GUI implementation, they choose the X Window System, which is a mature and stable product, and is also freely available for many computer platforms.

5.3.2 Toolkits

There are many toolkits available in X Window. The three products most widely available are MIT Athena, OSF/MOTIF, Open Look. Each of these products has a unique look and feel. As a result of OSF/MOTIF's [45] wide availability and completeness, it has been chosen as the graphic standard for the European Southern Observatory (ESO) [6].

An interesting development of OSF/MOTIF toolkit is its role as the basis of a proposed X/Open Standard:

> OSF/Motif has become the major Graphical User Interface (GUI) technology for Open Systems, as well as a de jure standard (IEEE P1295). The previous version of OSF/Motif (Release 1.2) introduced major new features such as internationalization, drag-and-drop and tear-off menus. Those features were intended to allow application developers to produce interoperable, easy to use applications for a worldwide market. As a result, this technology has been selected to become the basis of the Common Desktop Environment (CDE) jointly developed by HP, IBM, Novell and SunSoft, proposed to become a X/Open standard. [42]

With the availability of lesstif [37], the free source-compatible version of OSF/MOTIF. An even wider acceptance and use of the OSF/MOTIF standard can be expected.

5.3.3 GUI Builders

Adopting standards like X Toolkits (OSF/MOTIF) brings many benefits to the software developer. One important benefit which is not available in IRAF, is the availability of various GUI builder tools.

Two advantages of using GUI builder tools are:

- The quality of the interfaces might be higher. This is because the designs can be rapidly prototyped and implemented, even before the application code is written. Furthermore, different applications are more likely to have consistent user interfaces if they are created using the same GUI builder tool.
- The user interface code is easier and more economical to create and maintain, it is due to two reasons. Firstly, the separation of the user interface component from the application. Secondly, the GUI builder hides the complexities of the underlying system.

There are several public-domain GUI builders available for X Window development, like *Dirt*, *Serpent* and *xgen*. Some of the GUI in MIDAS have been developed with these public-domain GUI builders. Recently most of the GUI developments in MIDAS are done with a commercial package called UIDS (User Interface Design Systems) [6].

5.3.4 GUI Development

In a discussion session at a workshop, P. Ballester stated that it took about one week on average to add an OSF/MOTIF GUI to one package [6]. Examples of the packages are the XHelp package to access the online documentation, and XSpectra for reduction of long-slit spectra.

5.4 Comparison

The IRAF and MIDAS groups have chosen different implementations for their GUI. Therefore, the level of GUI integration and display ability are also different. A summary of the comparisons made is given in Table 5.1. Each entry in the table is discussed in the following:

Implementation - The IRAF Widget Server is theoretically sound. But in reality, this proprietary system is difficult to support as resources constraints are faced by programming groups in astronomy communities.

In contrast, MIDAS' choice of using standards like OSF/MOTIF and GUI builders is a sound decision. With OSF/MOTIF being part of the proposed X/Open standard, its future is assured. GUI builders allow developers to concentrate on design issues with rapidly prototyping, WYSIWYG (What You See Is What You Get), and spending less time on low-level implementation problems.

Display - The *ximtool*'s inability to display on a 24-bit display is unacceptable in today's computing environment. The programming group is unlikely to fix this problem any time soon, because of limited resources. MIDAS supports display depth of 8, 16, 24 and 32-bit.

GUI Integration - GUI in IRAF is supported by the external packages like *xgterm* and *ximtool* loosely tied with the system. The fact that IRAF comes without GUI support built-in shows how much work is still required to bring it to parity with MIDAS and IDL in GUI support. In contrast, MIDAS' GUI is built-in.

	IRAF		MIDAS	
	Comment	Score	Comment	Score
Implementation	IRAF Widget Server - hard to maintain	0	OSF/MOTIF - easy to maintain	1
Display	8-bit (*x11iraf*)	0	8/16/24/32bit	1
GUI Integration	Loose	0	Built-in	1
Overall	Poor	0/3	Good	3/3

Table 5.1: Graphical User Interface Comparison Summary

Chapter 6

Requirements and Installation

6.1 Requirements

The astronomy community has been a quick adopter of new technologies for the past few decades. The VAX/VMS was the dominant computer system for astronomers in the 1970s, and then it was Unix workstation during the 1980s. Today, many home PCs are more powerful than the workstation ten years ago, also very affordable.

Both IRAF and MIDAS run on the systems mentioned above, and more. Table 6.1 is a summary of the system requirements for both systems. From the table, it is evident that IRAF has higher system requirements than MIDAS. In the following sections, we will discuss each system's hardware and software requirements in detail.

6.1.1 IRAF Requirements

Included in this section is IRAF's hardware and software requirements from the "UNIX/IRAF Site manger's Guide" [63]. It should be noted that since IRAF was designed with networking ability in mind, it has a greater demand on the hardware and software in comparison with MIDAS.

Computer System

"Any modern host system capable of running Unix should be capable of running IRAF as well. IRAF is supported on all the more popular Unix platforms, as well as on other operating systems such as VMS.

"A typical small system is a single workstation with a local disk. In a typical large installation there will be one or more large central compute servers, each with several Gigabyte of disk and many Megabyte of RAM, networked to a number of personal or public workstations. For scientific use, a megapixel color screen is desirable.

Requirements	IRAF [31] 1999	MIDAS [16] 1999
Operating System	AIX, OSF, Ultrix HP-UX, IRIX, Linux FreeBSD, SunOS, Solaris VMS	AIX, OSF, Ultrix HP-UX, IRIX, Linux SunOS, Solaris, VMS
Memory		
Minimal	16Mb	8Mb
Recommend	128Mb	32Mb
Disk		
Minimal	500Mb	100Mb
Recommend	4Gb	200Mb

Table 6.1: System Requirements Summary

Memory Requirements

"The windowing systems used in these workstations tend to be very memory intensive; the typical screen with ten or so windows uses a lot of memory. Interactive performance will suffer greatly if the system pages a lot. Fortunately, memory is becoming relatively cheap. No system, including personal diskless nodes, should be configured with less than 32Mb of main memory; 48Mb or more is recommended if you plan to do a lot of image processing. On servers, 64, 128 or even 256Mb is not an unreasonable amount of memory to try to configure on the system.

Disk Requirements

"The amount of disk required by a user depends greatly on the application, so it is hard to recommend a minimum disk size. For a system with access to a central server, no disk or 200-300Mb of local SCSI disk is fine. For a stand-alone system with no access to large server, 500-600Mb is about the minimum. A server should have several Gigabyte of fast disk." [63]

Interestingly, the IRAF programming group has no intention to port IRAF to the DOS/Windows 95/98 operating system, as "it is an insufficient operating system to support IRAF" [31]. I share the same sentiment, as I believe that Unix systems are more robust, reliable and resilient than the DOS/Windows systems.

6.1.2 MIDAS Requirements

From Table 6.1, we can see that MIDAS has a definite lower system requirement than IRAF. I enquired Petra Nass of ESO whether the figures for MIDAS requirements are up-to-date, especially the 8Mb minimal memory requirement. Her reply is:

> 8Mb would still be O.K. - but this will also limit the size of the images you can work with. 8Mb is sufficient for 512 x 512 images. [43]

I believe the relatively low requirements of MIDAS has to do with its original design, which was for MIDAS to work on a non-networked single workstation or minicomputer in the early 1980s [17]. I have included in the following sections the suggested MIDAS' hardware and software requirements taken from the "ESO-MIDAS Requirements" on MIDAS' website [16].

Computer System

"MIDAS runs on VMS and OpenVMS systems, a large variety of Unix platform, as well as on personal computers running Linux. The availability for a specific system can be checked by asking the MIDAS Support.

Memory Requirements

"It depends on the number of users of the systems but normally at least 8 Mbyte. A physical memory that is too small may significantly reduce the performance due to swapping of data to disk.

Disk Requirements

"The full MIDAS system requires of the order of 20 Mbyte of disk storage if shared libraries (e.g., SUN, PC/Linux) are available, otherwise up to 100 Mbyte may be needed. During installation an extra 10 Mbyte should be available for temporary files such as object code. The size of the system can be reduced in three ways: a) source files can be deleted after implementation, b) help files can be removed if online help is not required, and c)parts of the system which are not used (e.g., crowded field photometry or echelle packages) need not be loaded. A typical disk size for a single user system is approximately 200 Mbyte assuming 60 Mbyte for the operation system, 80 Mbyte for MIDAS, and 50-100 Mbyte for a user with two-dimensional data." [16]

6.1.3 Comparison

A comparison of the hardware and software requirements is summarized in Table 6.2. Both systems run on a wide range of computing platforms and hardware, from the older VAX/VMS to the modern operating systems like Linux running on the latest hardware. Therefore, users should not have major problems getting IRAF or MIDAS running on their systems.

As mentioned in Section 6.1.2, MIDAS has a lower system minimum requirement than IRAF. Since the prices for memory and disk storage have decreased and increased in capacity, a common home PC today can easily meet the minimum requirements of both systems.

It should be noted that the user's requirements are different from the system minimum requirements. The system minimum requirements are the lowest requirements for the system to run in what its developers consider to be an acceptable speed. On the other hand, user's requirements are limited by factors such as size of the datasets to be processed, and users' financial constraints.

	IRAF		MIDAS	
	Comment	Score	Comment	Score
Operating System	Similar Coverage	1	Similar Coverage	1
Memory	Higher Demand	1	Lower Demand	1
Disk	Higher Demand	1	Lower Demand	1
Overall	Good	3/3	Good	3/3

Table 6.2: Requirements Comparison Summary

6.2 Installation

6.2.1 Introduction

In this section, I will explain the steps I have taken to install IRAF and MIDAS on to a PC running RedHat Linux 5.1. In brief, MIDAS is by far the easier package of the two to install if the user has the CD-ROM distribution.

I would characterize myself as experienced in installing software on Unix or Linux systems.

6.2.2 Testing PC Specification

I ran all the installation and testing on a Pentium 200MHz MMX PC, a modest computer system by today's standard. My choice of operating system was Linux, and I used the RedHat 5.1 CD distribution.

I have included the specification of this computer system in the following. The specification shows that it is an ordinary computer system that uses common computer parts. I observed no obvious hardware/software compatibility issues or performance problem when I ran IRAF and MIDAS on the system.

Hardware:

- ARTEC Pentium MMX Workstation
- Microstar Pentium ISO 9002 motherboard, Intel 430TX chipset, 512k pipeline cache
- Intel Pentium 200MHz MMX CPU
- 64Mb SDRam 10ns
- Quantum Bigfoot 4.3Gb EIDE hard drive
- Tseng Labs ET6000 4Mb 128bit 3D video card
- Wyse 17PS monitor
- 24x IDE CD-ROM
- 3COM 100base TX, PCI card
- Iomega internal 100Mb SCSI zip drive

Software:

- RedHat Linux 5.1
- NOAO PC-IRAF Revision 2.11.1
- ESO-MIDAS Release 98NOV pl2.1

6.2.3 IRAF Installation

Several astronomers have told me that even a basic IRAF installation is difficult. As a reasonably experienced Unix user, I find installing IRAF a tedious, though not difficult task. I will now describe the steps that I have taken to install IRAF on my PC.

The version of IRAF that runs on Linux is called PC-IRAF [61]. It was initially ported to Linux in September 1995. So far it supports Linux (RedHat, Slackware, and others) and FreeBSD, with ports to Solaris x86 and MkLinux in progress.

Briefly, in order to install IRAF on a PC, the user needs to follow the ten page PC-IRAF Installation Guide [33] that can be obtained from IRAF's web site. As I was unable to obtain the CD-ROM/tape distribution, I chose to download the distribution via anonymous ftp from *iraf.noao.edu*. The distribution does not include graphics and image visualization tools. Therefore, I downloaded the *x11iraf* package that can also be found in the directory */iraf/x11iraf*. Due to the difficulty of transferring large files over the internet, the gzip-compressed Tar files were split into smaller (512Kb) chunks to ease recovery from transfer failure. A total of 100 files (50Mb) were downloaded to my Linux box.

For administrative purposes, my next step was to set up a new account for the user *iraf*. After that I manually created the basic IRAF directory structure. For example, I created the *root*, *login* and *BIN* directories by typing the following commands:

```
% mkdir /iraf/iraf
% mkdir /iraf/iraf/local
% mkdir /iraf/irafbin
```

I extracted the gzip-compressed Tar files to their destined directories by typing the commands similar to the following:

```
% cd $iraf
% cat /home/ftp/pub/as.*  zcat  tar -xpf -
```

I then executed the PC-IRAF installation script to complete the system installation. The installation script modifies the host system to reflect the new root directory, new default image storage and local bin directories. Since the script also checks the mode and ownership of a number of files, installs a small set of IRAF commands in Linux, and so on. Therefore, root privilege is required to run the script to make the necessary modifications. Here are the commands I typed:

```
% cd /iraf/iraf/unix/hlib
% source irafuser.csh
% ./install
```

The basic system installation was complete after this stage, but I still needed to install the graphics and image visualization tools. I unpacked the binary archive for Linux, and had to manually move the binaries to */user/local/bin*, the app-defaults files to */user/lib/X11/app-defaults*, and the *man* pages to */user/local/man/man1*.

Lastly, before I could run IRAF, I had to run the *mkiraf* task to set up the IRAF login files, this completed the installation. To run IRAF, I executed the *cl* login script from my *root* directory:

```
% cl
```

During the installation, I had to refer to several READMEs and the Installation Guide for IRAF. I also made a few mistakes along the way. The installation took me just over 80 minutes to complete, not including approximately 40 minutes of downloading time.

6.2.4 MIDAS Installation

Paul Warhurst from the Physics department generously lent me his copy of ESO-MIDAS CD-ROM containing the release 98NOVpl2.1 (patch level 2.1) of ESO-MIDAS. I later also received a copy of the CD-ROM from Petra Nass at ESO.

Paul assured me that the installation of MIDAS is a breeze compare to IRAF. I proceeded to install MIDAS on my PC running Linux.

First thing I noticed was that a brief introduction to ESO-MIDAS, with instructions on installation, is included in the cover sheets of the CD-ROM. I browsed through the 15 pages of documentation in 2 minutes, and then turned to page 8 for instructions on installation from CD-ROM.

There are 2 compulsory steps and 3 optional steps in the installation procedure:

Step 1: Check disk requirement and create *MIDASHOME* directory

Step 2: Mount the CD-ROM and run the installation shell script

Step 3: Verify the installation (optional)

Step 4: Clean unnecessary packages (optional)

Step 5: Setup access to tape and printer devices (optional)

The minimum disk space required for a complete installation on a Linux machine (includes sources, binaries, demo data, calibration data) is about 144Mb. So I felt quite safe from running out of disk space on my 4.3Gb hard drive. Following the instructions, I logged in as *root*, created a home directory for MIDAS at */home/midas*, mounted the CD-ROM and executed the installation shell script.

A menu listing the steps in installing MIDAS was presented by the shell script. After I answered a few questions in each step, the script extracted MIDAS into the *MIDASHOME* directory (i.e., */home/midas*). Once the script completed the extraction, the basic installation was complete. It took me seven minutes to complete the two compulsory installation steps. To execute MIDAS, I can type *inmidas* in the shell to start a new MIDAS session, or *gomidas* to continue with a previous MIDAS session. Therefore my minimum installation of MIDAS took a total of nine minutes, excluding the time I spent acquiring the installation CD-ROM. I am impressed!

6.2.5 Comparison

MIDAS' easy installation made a big impression on me. It is almost as easy as installing software on a Macintosh. In contrast, IRAF's installation will be intimidating to inexperienced Unix users. There are six aspects of the installation that interest me: size of installation, time required for installation, type of media for distribution, user-friendliness and documentation. My detailed comparisons are summarized in Table 6.3. Below are justifications for each entry in the table.

- Size: Disk storage used to be a big issue in computing. However, with advanced storage technology, a 10Gb hard drive has become the norm for today's household PCs. A basic PC-IRAF installation takes up about 190Mb of disk storage, while MIDAS takes up 144Mb with all the calibration and demo data. In this regard, MIDAS has a slight advantage.

- Time: There is no contest here. Even without considering downloading time, it took me over 80 minutes to install IRAF. Comparing this to nine minutes for MIDAS, I will have to give MIDAS a full mark. It should be noted that since the software will be installed on a computer once only with infrequent updates, time should not be a big factor in the equation. It is good to be able to have the installation done in minutes instead of hours though.

- Media: Both software packages are distributed in various forms (CD-ROM, tape, network). Petra Nass from ESO kindly offered me a free copy of the CD-ROM distribution for MIDAS. Since I didn't have access to the IRAF CD-ROM, I downloaded the network distribution for IRAF from their ftp site. Unless the user has a fast internet connection, and a convenient way to download and manage 100 segmented files, CD-ROM will be the preferred option to the network distribution. The down side of CD-ROM is that it will be out-of-date when the new version comes out. On the other hand, the network distribution will always be up-to-date. So I see no reason in preferring IRAF to MIDAS, or vice versa.

- User-Friendliness: MIDAS' integrated installation script from its CD-ROM is preferable to the long series of Unix commands required to install IRAF.
- Documentation: Both software packages have well-written installation guides that can be obtained from their respective ftp sites or on the CD-ROM. The documents are in postscript or PDF format, so they can be viewed easily on the screen or printed out. It should be noted that the installation guide for PC-IRAF is a little bit out-of-date with the release of the latest version (v2.1.3) at the time of this writing. This caused some problems when I tried to install the latest version of IRAF on RedHat Linux 6.0 on my laptop.

	IRAF		MIDAS	
	Comment	Score	Comment	Score
Size	190Mb	1	144Mb	1
Installation Time	80 minutes	0	9 minutes	1
Media	Network	1	CD-ROM	1
User Friendliness	Poor	0	Excellent	1
Documentation	Good	1	Good	1
Overall	Fair	3/5	Excellent	5/5

Table 6.3: Installation Comparison Summary

One observation I made while preparing to install IRAF and MIDAS on my computer, is that neither system runs on Windows 95/98/NT. It is interesting that both projects decided to ignore the operating system that is running on 90% of the computers in the world today. As I noted in Section 6.1.1, the IRAF group explained its decision in the IRAF FAQ as follows: "it [Windows] is an insufficient operating system to support IRAF."

Chapter 7

Real World Experiences

7.1 Introduction

Paul Warhurst from the Physics department graciously accepted my request to perform a comparative experiment using both IRAF and MIDAS to reduce CCD images. The CCD images were taken for his photometric work on the star cluster NGC2516 with a visiting astronomer Michael Snowden. The experiment lasted two nights.

Paul is an expert user of MIDAS. He uses it frequently to reduce and analyze his CCD images. He is also a seasoned Unix user, who is familiar with the basic commands of the Unix system. Lastly he is a casual IRAF user.

We used IRAF and MIDAS to process the same dataset. The procedures that we took were similar to the procedures described in Figure 2.1. The basic three steps were: dark frame construction, flat field construction and image calibration.

In the following section, I will describe in detail the image reduction procedures using IRAF and MIDAS. It should be noted that the following is not a complete record of the sessions, but a summarized and idealized record of the sessions. Some trivial commands like copy files and change directory are not included. Also not included are the typing errors and minor mistakes we made.

7.2 IRAF

7.2.1 Preparation

At the beginning of the experiment, Paul and I spent several hours trying to get IRAF to read FITS (Flexible Image Transport System) format files, the format adopted by the astronomical community for data interchange and archival storage. The main difficulty we faced was finding documentation with instructions on setting IRAF to handle and manipulate FITS files in IRAF.

After we searched the IRAF website, it turned out that suitable instructions were included in the April 1998 IRAF Newsletter [12]. Later we realized the reason why we could not find the instructions in the IRAF documentation was that it was out of date with regard to the new IRAF extensions introduced in 1998 to support *imtype* and *imextn*.

Referring to the April 1998 IRAF Newsletter, we found that the correct way to set up FITS support in IRAF is to add the following lines in the file *login.cl* under the user's root directory:

```
imtype = fits
imextn = fxf:fts
```

We then added three lines to *login.cl* to force IRAF to preload the three packages we would be using to reduce the CCD images:

```
noao
imred
ccdred
```

One thing that we learned from trying to get IRAF to read FITS files is that IRAF cannot handle FITS files that have their file extension in capital letters. This is inconvenient for astronomers like Paul who use the DOS-based CCDOPS software for data acquisition at the telescope. The file names of the CCD images acquired by CCDOPS are converted to capital letters when transferred to the Unix-based system running IRAF. Therefore, after transferring the files to our Unix system, we converted the FTS extensions to lower case by using the following Unix pipelined command sequence:

```
% ls *.FTS | awk = F. '{print "mv "$0" "$1".fts"}' | sh
```

To enter the IRAF environment, we executed *cl* at the user's root directory:

```
% cl
```

7.2.2 Dark Frame Construction

The first thing we did was to create a combined dark frame. The combined dark frame was constructed to measure the dark current, which is the signal generated by thermal noise over the length of the exposure in the CCD chip. The dark frame would be subtracted from all the images taken by the CCD camera. We edited the parameters for combining the dark frames by the CL command:

```
cl> epar darkcombine
```

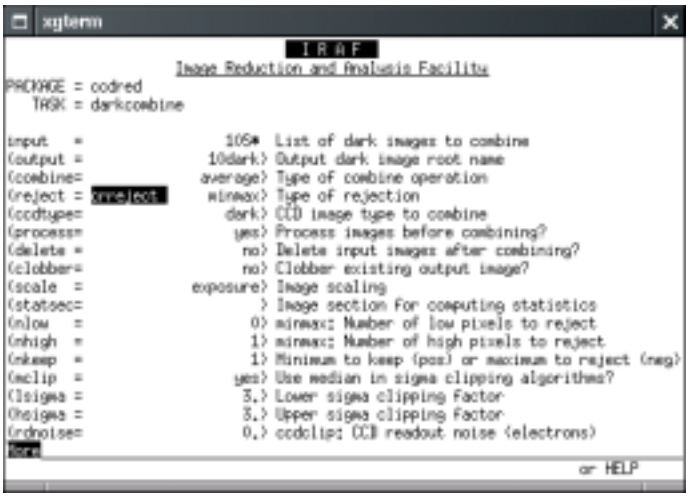

Figure 7.1: Editing Parameters in IRAF

IRAF called up an interactive screen editor with a list of parameters for darkcombine that can be edited interactively. Figure 7.1 is a screenshot of the interactive screen editor. We assigned values to the input frames, output frame, type of combine operation and pixel value rejection method. We finished editing by pressing CTRL+D to return to the CL prompt. We then executed *darkcombine* at the prompt and a combined dark frame was created which was called *10dark.imh*.

```
cl> darkcombine
```

Figure 7.2 is a dark frame of ten-second exposure, that was taken with the camera shutter closed, and displayed using IRAF's *ximtool*. Figure 7.3 is the dark frame that was formed by combining thirty dark frames, each of ten-second exposure to produce an averaged result.

Figure 7.2: Single dark frame displayed using IRAF's *ximtool*

Figure 7.3: Combined dark frame displayed using IRAF's *ximtool*

7.2.3 Flat Field Construction

A combined flat field frame similar to the construction of dark frame was constructed. The flat field is required to measure the relative sensitivities of every pixel on the CCD chip.

Before we could combine the flat fields, we had to subtract the dark frame from every flat field frame. This was achieved by using the *ccdproc* routine. We had to edit the parameters to specify the dark frame to be used, which is *10dark.imh*. Then we invoked the *ccdproc* routine:

```
cl> epar ccdproc
cl> ccdproc
```

We edited the flatcombine parameters using an editing screen similar to that shown in Figure 7.1. Finally, we invoked the *flatcombine* routine by the following commands:

```
cl> epar flatcombine
cl> flatcombine
```

7.2.4 Image Calibration

The last stage of image reduction was to subtract the dark frame from the raw CCD image and flat field the image. We used the *ccdproc* routine to perform this task. We edited the parameters to specify the dark frame and the flat field to be used, and invoked the *ccdproc* routine:

```
cl> epar ccdproc
cl> ccdproc
```

To examine the now calibrated image, we displayed the image by invoking *ximtool*, the external display server. The exclamation mark in front of command meant that it was a shell command. We then displayed the image with the *display* command. It created an FIFO pipe between IRAF and *ximtool*, and the image *F1R001.imh* was displayed in the *ximtool* window.

```
cl> !ximtool
cl> display F1R001.imh
```

7.2.5 Summary

The following is a summary of the issues arisen from the experiment with IRAF:

Parameter Editing - IRAF's interactive screen editor for editing parameters is well designed. A list of the parameters is displayed on the screen, and the user can edit them interactively. The screen editor also checks for valid entries, and provides a list of valid options if the entry is invalid. It was a very useful tool for us during the experiment.

Command History - The command *history* (or *his*) is used to display the command history, and the command *ehistory* (or just *e*) is used to access previous commands. From our experience, the command history mechanism is not well integrated into the command line interface, and the interface is less intuitive than MIDAS.

Command Minimum Matching - The IRAF command minimum matching is discussed in detail in Section 3.3.2. There was an incident during the experiment in which Paul typed *log* at the IRAF prompt, and I thought he was going to check out the system log. Instead of printing out the log, he logged out from the IRAF environment, because the word *log* is a unique match to the command *logout*. To my mind, this is a confusing user interface design.

Documentation - It was a frustrating experience for us not to be able to find out how to set up *imtype* and *imextn* from IRAF's documentation. Both the "A Beginner's Guide to Using IRAF" (1993) and the "IRAF Tutorial" (1997)

were last updated before the introduction of *imtype* and *imextn*. Obviously it is not practical to update the documentation whenever a new version comes out, but I believe important information like how to set up *imtype* and *imextn* should definitely be included in the beginner's guide and the tutorial.

Image Display and Manipulation - IRAF's *ximtool* is a robust image display server packed with features like real-time brightness/contrast adjustment, adjustable blinking, and a GUI interface. It was a very useful utility that we used to examine the images during the experiment. Regrettably, the user has to start *ximtool* manually, and it only supports 8-bit PseudoColour displays.

7.3 MIDAS

7.3.1 Preparation

We invoked *inmidas* in the shell to start a new MIDAS session. We then had to enable *ccdred*, the CCD Reduction context for MIDAS. Contexts are special applications that are available as individual packages inside MIDAS. The context was enabled by the following command:

```
% inmidas
Midas 001> SET/CONT ccdred
```

7.3.2 Dark Frame Construction

We had to create catalogues in MIDAS to work on multiple images simultaneously. The following command creates a catalogue of thirty dark frames, and each ten-second exposure that we wanted to combine:

```
Midas 002> CREATE/ICAT 10cat 10sec00*
```

Before we can combine the 30 dark frames, we had to set a few parameters. We used the following commands:

```
Midas 003> SET/CCD DK_MET=mean
Midas 004> SET/CCD DK_SCA=yes
```

We executed the following command to create *dark10.bdf*, a combined dark frame image. This combined dark frame would be used in a process which astronomers called "Dark Subtraction" on the other CCD images.

```
Midas 005> COMBIN/CCD dk 10cat.cat dark10
```

7.3.3 Flat Field Construction

First, we had to create a catalogue of the flat field images to be combined:

```
Midas 006> CREATE/ICAT rcat RFL00*.bdf
```

Before we could combine the flat fields, we had to subtract the dark frame from every flat field frame. This was done by executing the following command:

```
Midas 007> EXECUT/CAT dark/ccd rcat.cat rcat.cat dark10
```

To combine the flat field, we had to set a few parameters. First we examined the parameters for combining the flat field by the following command. A screenshot of the parameter list is provided in Figure 7.4.

```
Midas 008> SHOW/CCD ff
```

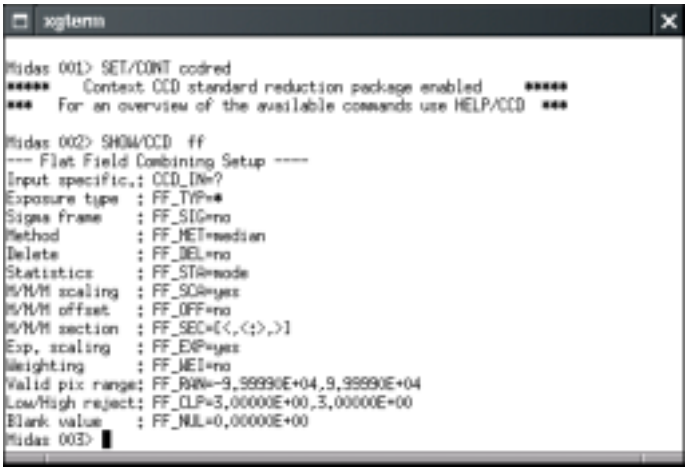

Figure 7.4: List of Parameters for Combining Flats in MIDAS

Then we changed the values of the two parameters *FF_MET* and *FF_EXP*, and combined the flat fields by the following commands. Whose resulting image has named *flatred.bdf*.

```
Midas 009> SET/CCD FF_MET=mean,FF_EXP=no
Midas 010> COMBIN/CCD ff rcat.cat flatred
```

Figure 7.5 is a flat field of ten-second exposure displayed using MIDAS. Figure 7.6 is a combined image of five flat field, each of ten-second exposure.

7.3.4 Image Calibration

Again, we had to create a catalogue of the list of CCD images (with filenames of the form *toto000*.bdf*) that we wanted to process:

```
Midas 011> CREATE/ICAT totcat toto000*
```

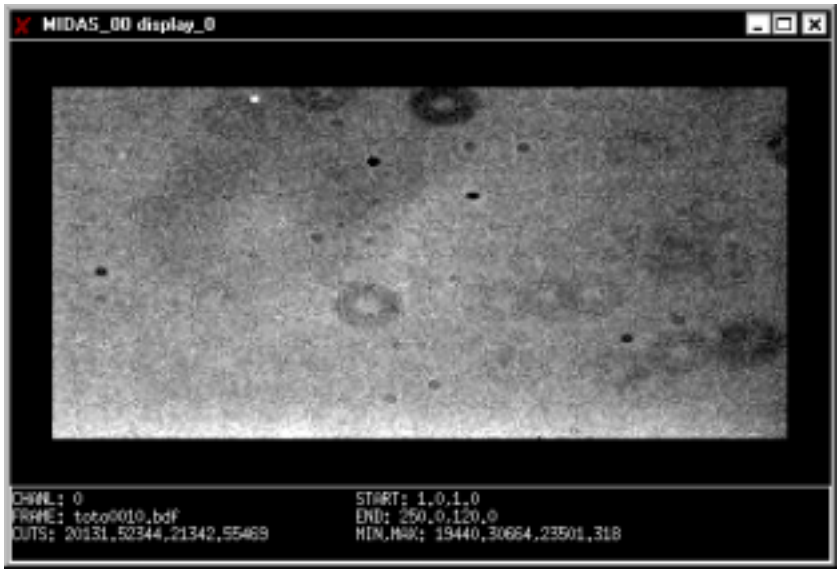

Figure 7.5: Single flat field image displayed in MIDAS

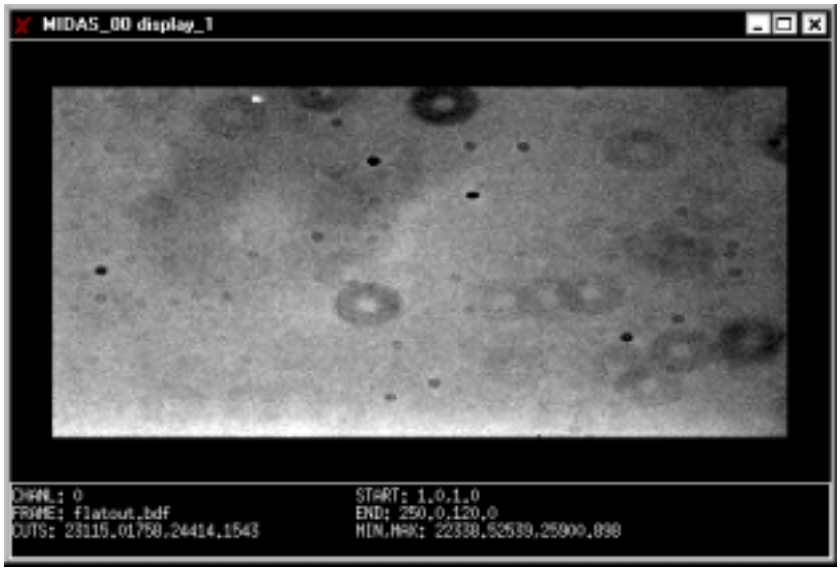

Figure 7.6: Combined flat field image display in MIDAS

The dark frame was subtracted from the list of CCD images:

```
Midas 012> EXECUT/CAT dark/ccd totcat.cat totcat.cat dark10
```

The CCD images were flat fielded by executing the following command:

```
Midas 013> EXECUT/CAT flat/ccd totcat.cat totcat.cat flatred
```

To display the calibrated image, we executed *LOAD/IMAG*. The image, in this case *toto0001.bdf*, was displayed in a window created by MIDAS, enlarged by 200%, and with a cut value at three sigma.

```
Midas 014> LOAD/IMAG toto0001.bdf scale=2 cuts=f,3
```

The effects of dark current subtraction and flat fielding are clearly demonstrated in Figure 7.7, Figure 7.8 and Figure 7.9. The raw CCD image of a section of the star cluster NGC2516 seems to be carrying a lot of noise in Figure 7.7. Figure 7.8 shows a dramatic improvement in signal-to-noise ratio after dark current has been subtracted from the raw image. Lastly, Figure 7.9 shows the image after the field has been flattened.

7.3.5 Summary

Here are some observations I made during the experiment with MIDAS:

Parameter Editing - Compared with IRAF, it is less intuitive for the user to access and modify command parameters in MIDAS. The user has to list the command parameters, then look at the list to decide what parameters are to be changed. Only then can the user set the new values of the parameters one by one with the SET command. The user will not know if the value

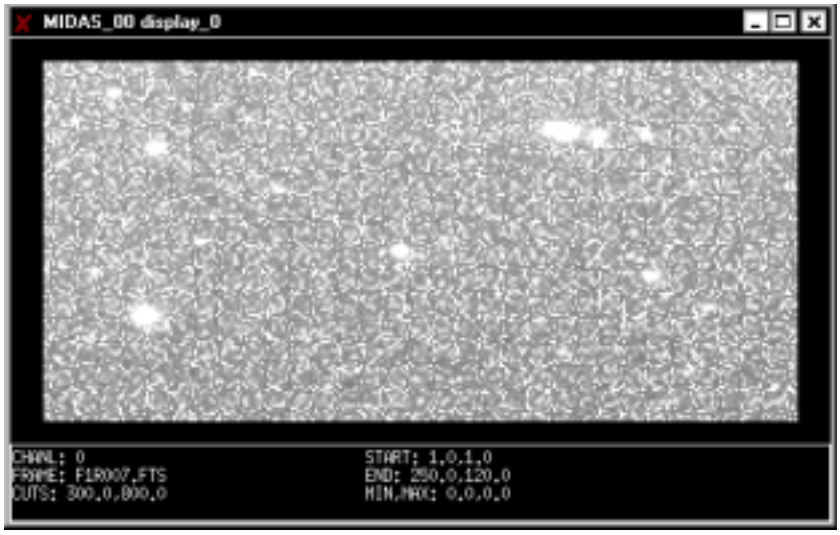

Figure 7.7: Raw CCD image of a section of the star cluster NGC2516

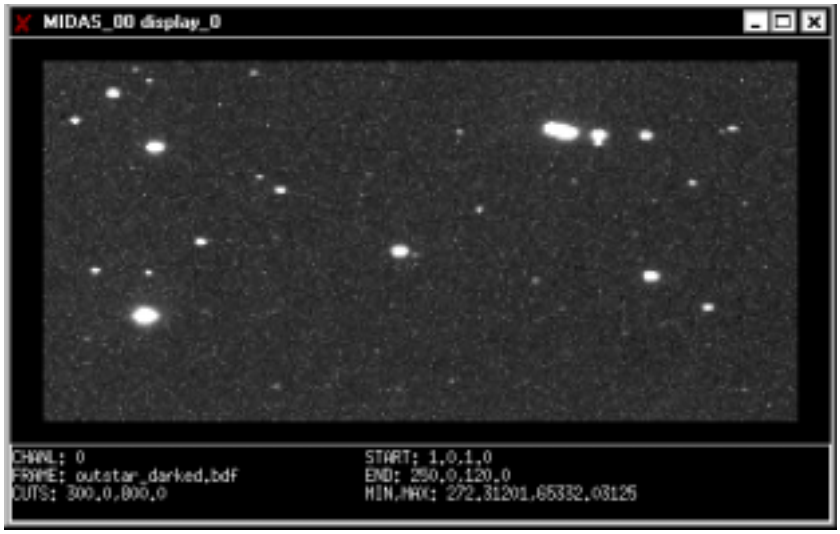

Figure 7.8: Dark subtracted frame of the same CCD image

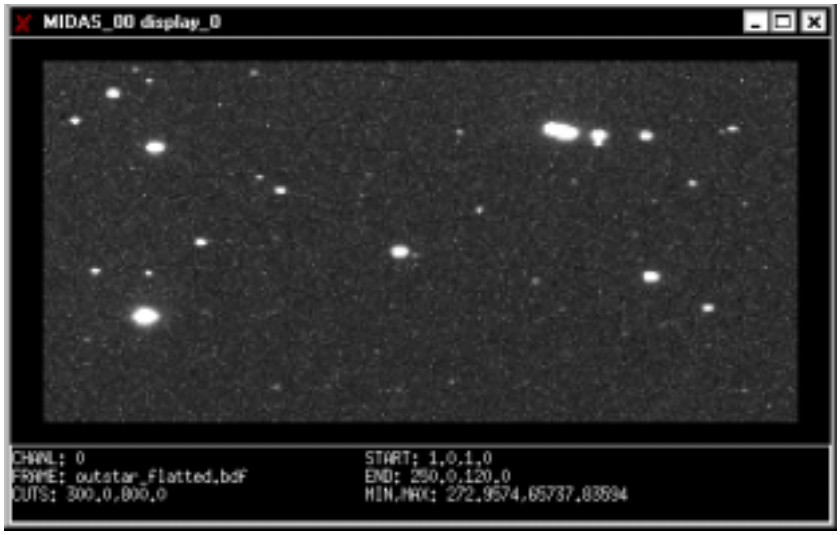

Figure 7.9: Dark subtracted and flat fielded frame of the same CCD image

of the parameter is valid until he/she executes the task. Furthermore, it is more likely for users to mistype parameter names in MIDAS than in IRAF.

Command History - Command history and numbering are fully integrated into MIDAS command line interface. Each command line prompt is numbered, so the user can simply type the number to recall the command. Moreover, the user can display the command history simply by pressing the ENTER key. Lastly, arrow keys are used to recall and edit previous commands.

Command Abbreviation - The convenience of command abbreviation in MIDAS cannot be emphasized enough. We used it all the time during the experiment to let MIDAS auto-complete the incomplete command names, file names, and path names. Both Paul and I identified command abbreviation as the most important feature from our experiment. A detailed discussion on command abbreviation is in Section 3.3.2.

Image Display and Manipulation - MIDAS' image display and analysis mechanism is an integrated part of the MIDAS environment. It is very easy to create windows which display images or graphs. All the commands are entered via the command line interface, with some mouse interaction. In my opinion, GUI interface is a better interface which provides easy access to image display/manipulation commands and parameters.

7.4 Comparison

There are four issues that I noted as important during the experiment: parameter editing, command history, command abbreviation and image display/manipulation. Table 7.1 is a summary of the issues which are discussed in the following:

Parameter Editing - IRAF's interactive screen editor for editing parameters was very useful in our experiment. In contrast, it was a very troublesome task to set parameters in MIDAS.

Command History - Both IRAF and MIDAS feature command history and command numbering with subtle differences in implementation. In my opinion, the differences in the implementation do make a noticeable difference to the user experience. In my opinion, MIDAS has a better interface to command history than IRAF.

Command Abbreviation - This point has been discussed in detail in Section 3.3.2. In brief, we found that MIDAS' command abbreviation facility worked brilliantly, while we found it frustrating to use IRAF's minimum matching feature.

Image Display and Manipulation - Both systems have their advantages and disadvantages. I believe that improvements can be made to IRAF by tightening the integration between the system and its display server. Furthermore, MIDAS needs a graphical user interface (GUI) for its image display and manipulation functions.

My overall impression of this experiment is that both packages get the job done efficiently, nonetheless, MIDAS does create a better user experience than IRAF.

	IRAF		MIDAS	
	Comment	Score	Comment	Score
Parameter Editing	Good	1	Average	0
Command History	Average	0	Good	1
Command Abbreviation	Average	0	Excellent	1
Image Display and Manipulation	External	1	Built-in	1
Overall	Fair	2/4	Good	3/4

Table 7.1: Real World Comparison Summary

Chapter 8

Current Status and Future Plans

8.1 IRAF

8.1.1 Current Status

From my private correspondence with members from the IRAF group, it is their belief that "the software has been remarkably stable for the last ten years." [62] They also believe "the original architecture and interfaces are still very sound." [20] It is my opinion that IRAF may be is a stable system, but its software architecture and interfaces are starting to become out of date. In the next section, I will discuss the IRAF group's plan to take IRAF to the future.

The focus of recent IRAF development has been concentrated mainly on developing applications of all sorts, from data acquisition to support for various new instruments. In the system software area there have been a GUI development project, an addition of FITS support, and the release of X-based image display tools such as *xgterm* and *ximtool*. Moreover, IRAF was ported to run on a PC under Linux/FreeBSD in 1998 [61].

The IRAF group has ten staff at the time of this writing. Even though it is impossible to keep an accurate count of the number of users and sites, IRAF's records suggest an estimated 5,000 users at 1,500 sites around the world. All of them use IRAF to perform image reduction and analysis [31].

From the number of recent papers presented in ADASS, it is clear that a great amount of development has been done and continues to be done in IRAF and its applications. This development is not only occurring in the IRAF group, but also in other research institutions around the world.

8.1.2 Future Plans

I asked the designers and programmers from the IRAF group about the future of the IRAF project. To my knowledge, there is so far no plan to rewrite IRAF from the ground up, analogously to the way AIPS is being re-engineered into AIPS++ (see Section 9.3. My IRAF respondents estimate that it would take another ten years to rewrite IRAF, given their very limited resources. Any rewriting of IRAF would adversely affect other projects like STSDAS (Space Telescope Science Data Analysis System) that builds on top of IRAF architecture [20].

However, there is a new multi-year project called OpenIRAF. The goal of this project is to modernize IRAF by providing language bindings for existing interfaces (e.g., FORTRAN, C, C++, Java), with an open architecture to allow other systems (e.g., IDL) to be used within IRAF environment [60].

In the long term, a major new system will be developed by the IRAF group. However, it will not be developed from scratch, but in parallel with the existing IRAF system. The group is quietly developing the new system framework which is based on distributed objects, components, and messag-

ing. When the framework is fully developed, the tasks in the original IRAF can be reused by repackaging, and then be moved to the new framework. The new applications will be written as components. These components will be linked by an interpreted language to form the new IRAF environment [62].

8.2 MIDAS

The current status and the future of MIDAS are less certain. Even though MIDAS is still slowly being maintained and updated to run on the latest operating systems, most of programming effort went into the development of the VLT pipeline six or seven years ago [66]. To my knowledge, no clear plans or time-scale for MIDAS' development have been set [43]. Still, in a 1990 article titled "Acquisition, Processing and Archiving of Astronomical Images" [39], we get a glimpse of the major new design features which could have been added to the MIDAS system over the past decade:

Network database - Access to data file over a network includes machines with different internal design.

Distributed System - The possibility to execute a MIDAS task on different nodes in a network. This is needed to share central computers between many interactive users on workstations.

Adaptive User Interface - Different user interfaces which depend on the application.

Data Acquisition - Integrating links to data acquisition processes needed for interaction with remote observing and usage as a close to online system.

MIDAS is distributed under the GNU General Public License (GPL). Therefore, the source code is available to developers or institutions interested in modifying or providing new functionality.

8.3 Comparison

Both the IRAF and MIDAS group have recently released new versions of their respective software. These new updates are mainly responsible for bringing compatibility to new operating systems, and patches for bug fixes.

From the number of recent papers presented in ADASS related to IRAF, it is evident that IRAF has been actively developed by both the IRAF programming group and other institutions around the world. It is my belief that IRAF has more features and applications than any astronomical software packages available on the current market.

Even with limited resources, the IRAF group is moving on with new initiatives like OpenIRAF and development of new system framework as I have described in Section 8.1.2.

MIDAS' development has been held back in recent years, as a result of the shifting of programming staff from MIDAS to VLT's data pipeline development, which is part of ESO's VLT Data Flow System (DFS) [65], six years ago. I think that the development of DFS is very exciting for the users of the VLT telescopes. But it has fewer direct benefits to the general astronomy community that uses MIDAS for data reduction and analysis.

With the uncertain future of MIDAS, it is my opinion that the institutions and individual users who use MIDAS as their primary data reduction and analysis package should consider forming an international consortium to continue the development of MIDAS. There is already an example in the astronomy community that an international consortium was formed to carry on the development of a software project started by one institution. Here I am referring to the AIPS/AIPS++ project, which will be discussed in detail in Chapter 9.

A summary of the above comparison is in Table 8.1.

	IRAF		MIDAS	
	Comment	Score	Comment	Score
Current Status	Good	1	Poor	0
Future Plans	Good	1	Poor	0
Overall	Good	2/2	Poor	0/2

Table 8.1: Current Status & Future Plans Comparison Summary

Chapter 9

Other Astronomical Software Packages

9.1 Introduction

This chapter is an introduction to three alternative software packages for astronomy. AIPS and AIPS++ are open source software packages mainly used for radio astronomy, while IDL is a commercial software development environment used extensively in space astronomy. Since the focus of the thesis is on inexpensive software packages for optical astronomy, this chapter is devoted to the discussion on these software packages.

9.2 AIPS

The Astronomical Image Processing System (AIPS) [3] is developed by the National Radio Astronomy Observatory (NRAO). It is developed for interactive (and, optionally, batch) calibration and editing of radio interferometric data and for the calibration, construction, display and analysis of astronomical images made from those data using Fourier synthesis methods.

The design and development of AIPS began in Charlottesville, Virginia in 1978. Since then, approximately 50 person-years of effort has been put into the development and documentation of AIPS. The principal users of AIPS are VLA, VLBA, and VLBI Radio Telescope Network observers. VLA (Very Large Array), VLBA (Very Long baseline Array) and VLBI (Very Long Baseline Interferometry) are some of the biggest radio telescopes in the world. It is interesting to note that in a survey carried out in late 1990 showed that 7% of AIPS registered sites were using AIPS for non-radio work [9].

AIPS is written in the FORTRAN programming language. Object-based FORTRAN was introduced to AIPS in 1992 as the "OOP" package, to explore the advantages of object-oriented methodology. It provides easier FORTRAN access to data structures, and operations on entire data structures [64].

Due to the new AIPS++ project, the development of AIPS has been limited to general code maintenance and improvements in documentation. The group is now called the "Classic AIPS group" to distinguish itself from the AIPS++ project. At the time of this writing, the latest version is the 31Dec99 AIPS.

9.3 AIPS++

9.3.1 Introduction

Realizing the potential benefits of object-oriented techniques, a consortium of seven observatories from around the world (Australia, U.S.A., Canada, Netherlands, U.K., India) was set up in 1992 to start a new project called AIPS++ [4]. The new project seeks to replace all the functionality of original AIPS using modern coding techniques (OOP) and languages (C++). It is currently the largest strictly object-oriented post-processing software effort in the astronomical community [22].

My understanding of the original goal of the AIPS++ project, gained from private correspondence with Tim Cornwell, the AIPS++ project manager, is to create a modern data reduction package that is capable of reducing data acquired from the two radio telescopes VLA and VLBA [10].

In "Creating an Object-Oriented Software System - The AIPS++ Experience", Brian Glendenning, a former AIPS++ member, now the head of Software Working Group for the Atacama Large Millimeter Array (ALMA) project summarized the status of the AIPS++ project in 1996:

> AIPS++ is a controversial project. Much of the controversy stems from the twinned observations that AIPS++ is being implemented with new techniques, and that the AIPS++ project is running considerably behind the schedule it originally promised. [22]

In the following sections, I will discuss three important issues arisen from the AIPS++ project: Object-Oriented Programming (OOP), revolution vs. evolution, and the choice of the Glish scripting language. We will explore the difficulties faced by the project during its development. Furthermore, the issue whether the project is worth the huge amount of effort and resources invested by the members of the international consortium.

9.3.2 Object-Oriented Programming

In 1996, after several years of development, Glendenning summarized the experience of the AIPS++ development team with OOP and C++ as follows:

Reliability - Encapsulation of objects is functioning well in keeping bugs isolated. The bugs are relatively easy to find and fix.

Reusability - By policy, AIPS++ is unable to use commercial libraries because the software must be freely redistributable. However, the internal reuse of the class libraries they have developed has worked very well.

Decoupled development - This has worked well once the design was finalized and the foundation classes were stable.

Productivity - AIPS++ was about two or three years behind schedule few years ago. Nonetheless, as the programmers have climbed the learning curve and overcome technological problems (e.g., the standardization of C++), the project is presently on schedule. [22]

All in all, Glendenning believes that AIPS++ has greatly benefited from using object-oriented technology and C++, and this benefit will compound in time. My opinion is that the designers of the AIPS++ project were overly influenced by the Object-Oriented Programming and C++ hyperbole of early 1990s. I am glad that they have overcame the problems that they have faced and climbed the steep learning curve of OOP and C++. Nevertheless, I share Tim Cornwell's opinion that OOP and C++ was not needed for a project like AIPS++ that was well-defined and limited in goals [10].

9.3.3 Revolution vs. Evolution

The AIPS++ consortium has chosen the revolutionary approach, that is, rewriting the entire system. The obvious alternative is an evolutionary approach. One reason to adopt a revolutionary approach is that the cost is lesser make changes all at once rather than integrating many major changes into the system over time. It is very hard to incrementally make fundamental changes, since fundamental design decisions have far-reaching implications in a software system [22].

Tim Cornwell expressed his opinion on the revolution vs. evolution debate from the perspective of the AIPS++ project manager:

> The revolutionary approach has its advantages. A fresh start enables much better techniques to be used but at the cost of political support. But in an overly revolutionary approach, support from managers is hard to keep since nothing visible happens for a long time...
>
> An evolutionary approach is good for political support but would compromise the eventual deliverables. [10]

A brief discussion on the IRAF group's choice of an evolutionary approach to the development of IRAF is included in Section 8.1.2.

9.3.4 Scripting Language - Glish

It is the AIPS++ group's opinion that mastering all or most of C++ is a considerable undertaking for their target users. Therefore, the AIPS++ group hope that the end-users and programmers will use the AIPS++ scripting language, Glish, to program their ad hoc calculations and algorithms.

Glish was primarily developed at the Lawrence Berkeley Laboratory (LBL), and much of the initial development was funded by the Superconducting Super Collider project [46]. The AIPS++ group has taken over and extended the Glish's features from 1995.

Glish is a language (loosely) based on the language *S* used by statisticians. It has a C-like syntax, powerful whole-array primitives, a simple event-driven mechanism for incorporating functionality from external binaries. Moreover, a direct binding to *Tk* widgets allows easy creation of custom GUI's. The developers of AIPS++ believe that Glish, when combined with functionality in the AIPS++ library, enables the user to create useful ad-hoc calculations without much difficulty [54].

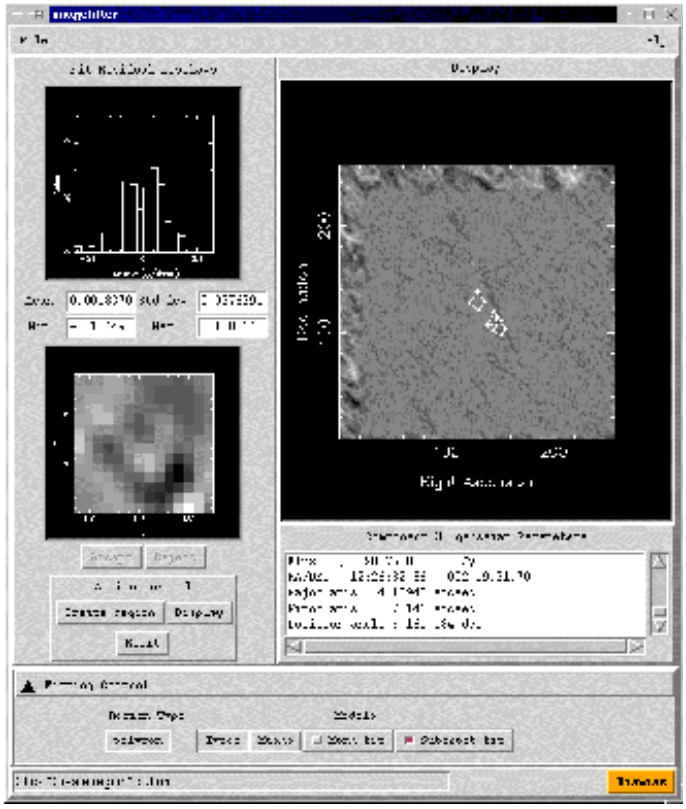

Figure 9.1: AIPS++ Application: Image Fitter - An interactive and batch image plane component fitter. (Curtsey of the AIPS++ Group)

As I have discussed in Section 3.3.1, there is a trend in the astronomical programming community to adopt the design of the modern general purpose scripting languages. AIPS++ has this advantage by using Glish, when compared to IRAF and MIDAS' aging scripting languages.

9.3.5 Current Status & Future plans

This section is an overview of the status and plans of the AIPS++ project gained through private correspondence and my analysis of the published literature.

According to the AIPS++ project manager, Tim Cornwell, the AIPS++ project is presently on schedule, after several years of lagging behind [10]. At the moment, there are about 20 people working in the AIPS++ consortium [68]. The first public release version of AIPS++ (version 1.2) was released in October 1999, and the consortium is scheduled to make releases every 6 months. The released version of AIPS++ package is freely distributed in the form of CD-ROM on request [44].

An interesting development of AIPS++ is the parallelization of AIPS++ at the National Center for Supercomputing Applications (NCSA) at University of Illinois at Urbana-Champaign. It uses the standard AIPS++ User Interface with the parallelization infrastructure overlaying on existing AIPS++ algorithm implementation. The implementation of parallel processing uses the Message Passing Interface (MPI), which is a portable system that allows data and instructions to be sent to remote processors. Preliminary testing shows good speedup (linear) on systems with up to 15 processors [69].

According to Dr. Cornwell's prediction, it would take about another year of development for AIPS++ to reduce data acquired from the radio telescopes VLA and VLBA, which is the original goal for AIPS++. However, AIPS++ has been used at a number of places for mission critical work. The GBT (Green Bank Telescope) is being commissioned to use AIPS++. NRAO is setting up a Data Management Division to use AIPS++ for automated data reduction for NRAO's telescopes. Lastly, the AIPS++ project has just won a substantial NSF (National Science Foundation) grant for their work on visualization in AIPS++ [10]. In short, I agree with Dr. Cornwell that the prospects for AIPS++ are very good, especially when compared with IRAF and MIDAS that both groups are under tight resource constraints.

9.4 IDL

9.4.1 Introduction

The Interactive Data Language (IDL) is a commercial scientific computing environment by Research Systems, Inc. (RSI). It combines data analysis, 2D/3D visualization of data, mathematics and statistics libraries, integrated development environment and cross-platform GUI tools in one package. IDL is used extensively in space astronomy. It is being used as the main language to develop reduction packages for missions like IUE (International Ultraviolet Explorer), HST (Hubble Space Telescope), ROSAT (ROentgen SATellite), SOHO (SOlar and Heliospheric Observatory) etc. It runs on all the major computer platforms (PC/Mac/Unix/VMX) [48].

9.4.2 Brief History

A brief history of IDL development was given in the IDL frequently asked questions [29]. I have extracted and summarized it as the following:

IDL is a product of Research Systems, Inc., founded by David Stern in 1977. It was originally developed at the Laboratory for Atmospheric and Space Physics (LASP) at the University of Colorado.

During his time at LASP, David Stern conducted a research on making computers more user-friendly for the physicists at the Lab. The first of the two programs that led to the development of IDL was Rufus. Rufus was a simple vector oriented calculator that ran on the PDP-12. A later version was the Mars Mariner Spectrum Editor (MMED) that ran on the PDP-8. The second program was named SOL. The design of SOL was influenced by the APL language, it was array oriented and had some primitive graphics capabilities. APL (A Programming Language) is a mathematically inspired array oriented language created in 1960s at IBM to serve as a powerful executable notation for mathematical algorithms.

David Stern left LASP in 1977 and started RSI with the intention of expanding the ideas contained in SOL. The initial result was PDP-11 IDL which was much more capable than SOL. Later IDL was ported to VMS, various Unix systems and now Microsoft Windows and Macintosh.

9.4.3 Features & Limits

The following is a list of features/limits of IDL summarized from my web search on comments made by IDL users. The major features/merits of IDL that are of interest to an astronomer are:

- The array oriented nature of IDL means arrays may be referenced without the use of subscripts or loops, and the code is optimized for array operations. Therefore, it appeals to scientists and engineers in the analysis of one, two or three-dimensional datasets.
- IDL code is mostly site-independent, this means that the code can be executed with only minor modifications for the local configuration in most situations.
- The IDL Astronomy User's Library is an archive of 300+ free astronomically oriented routines. These utilities can be readily used in many data reduction and astronomical problems. The source code is also available, so the user can use them to write, customize or to correct errors.
- The ability to easily and quickly create widget tools, and to constantly modify them according to changing requirements are considered as the most important advantages of IDL [55].
- IDL GUIBuilder allows drag-and-drop cross-platform GUI development, which is fully integrated within the IDL Development Environment (IDLDE). It is currently available for computers running the Windows operating system only.

My web search also revealed some concerns about the limitations of IDL:

- It is a proprietary system which means that each site must purchase an IDL license. Astronomers will want to use the system at home and the cost of another license may be prohibitive for most university-based observers.
- Native IDL is not as thoroughly tested as FORTRAN or C. Therefore, some bugs are inevitable.
- The object-oriented facilities are recent add-ons and have their limitations – accessing an object's properties is clumsy, and there is no operator overloading as there is in Matlab [24].
- The use of virtual memory makes IDL unsuitable for large images ($\geq$ 16Mb) or several images at once. Large images must be segmented to be processed with IDL [48].
- Software developed with IDL can only be exported to other IDL licensed sites, a point already discussed in detail in Section 1.2.3. IDL is currently developing an exportable execution format to overcome this limitation [30].

9.4.4 Summary

My impression of IDL is that it is an excellent software development environment for scientific research. Unfortunately, it fails as a ready-to-use astronomical image reduction package. With the free distribution of major astronomical image reduction and analysis packages (IRAF, MIDAS, AIPS++), IDL must offer significant and obvious advantages to overcome the price difference. One advantage of IDL is its ability to run on the Windows 95/98/NT platform, while others cannot.

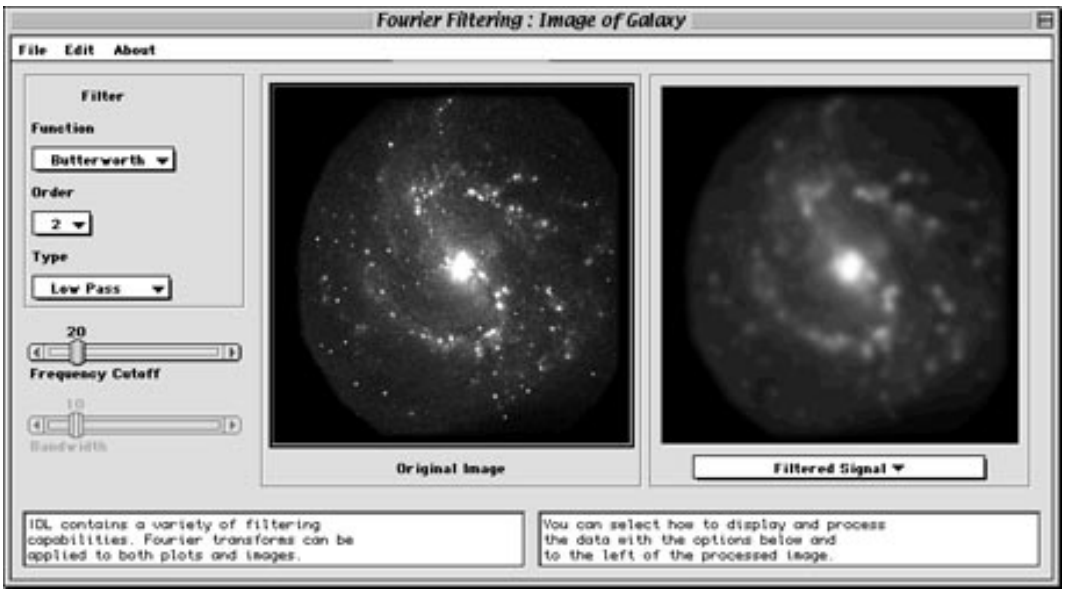

Figure 9.2: IDL Fourier Filtering Demo Screenshot

Chapter 10

Conclusion

10.1 What have we found?

The aim of this thesis is to carry out a comparative study of IRAF and MIDAS to help prospective users decide which software suits them best. The following is a summary of the main points of this thesis:

Chapter 1 - So far, there is no ideal software package for observational astronomers on the market.

Chapter 2 - Both IRAF and MIDAS were developed right after the CCD revolution for observational astronomy in the early 1980s. These software packages were developed to aid astronomers in the reduction of images taken with the CCD camera mounted on a telescope. Reduction is a process of turning raw CCD images with background noise into calibrated images.

Chapter 3 - IRAF and MIDAS have similar software architecture design, but are very different in their implementation. MIDAS' command abbreviation is much easier to use than IRAF's command minimum matching mechanism.

Chapter 4 - The major difference between the IRAF and MIDAS application programming environments is the choice of programming languages. SPP is the choice for IRAF, and FORTRAN and C for MIDAS. MIDAS has a better programming environment, as FORTRAN and C are popular programming languages, with many tools and utilities available.

Chapter 5 - IRAF is strong in terms of functionality, but is weak in the area of user interfaces. External utilities, like *xgterm* and *ximtool* were developed using the IRAF Widget Server to provide GUI for IRAF. MIDAS has a built-in GUI which uses the popular OSF/Motif toolkit.

Chapter 6 - Both IRAF and MIDAS have very similar hardware/software requirements. Interestingly, Microsoft's Windows 95/98/NT operating systems are not supported. The MIDAS installation procedure is easy, while it requires experience and patience to install IRAF.

Chapter 7 - In an experiment conducted with a local astronomer, we found that both packages got the job done, but MIDAS offered a better user experience than IRAF.

Chapter 8 - Even with limited resources, the IRAF group is moving on with new initiatives like OpenIRAF and new system framework development. MIDAS' future is less certain, as most of programming effort started to go into the development of VLT pipeline six years ago.

Chapter 9 - There are currently many alternative astronomical software packages available for professional and amateur astronomers. Three of these alternatives were discussed briefly in this thesis: AIPS, AIPS++ and IDL. They were not discussed in depth, as the focus of this thesis is on inexpensive software packages for optical astronomy.

I have presented a vast amount of information on IRAF and MIDAS in this thesis. The questions is how can a prospective user make efficient use of so much information to determine which software meets their criteria? In the following section, I will describe Multiple Criteria Decision Analysis. This should help prospective users make a better decision based on their particular wants and needs.

10.2 Multiple Criteria Decision Analysis

To make a quality decision, a clear strategy should be applied in the decision making process. Decision strategies can be broadly classified into compensatory and non-compensatory classes. Dr. Bruce Landon from the Psychology Department, Douglas College, Canada states:

> Non-compensatory strategies eliminate alternatives by removing alternative candidates that have a critical flaw on any of the decision criteria...
>
> The other major approach to handling complex decision is to use a compensatory strategy that allows for stronger aspects of a candidate to make up for other weaker aspects of the candidate. Conceptually it is very similar to the conventional weighted grading approach... [36]

I have chosen to use a simple weighted grading method for Multiple Criteria Decision Analysis. This compensatory method is based on the idea of continuous tradeoffs between value attributes. The increase of value in one attribute can compensate for the decrease in another. My technique is described in the following:

1. Generate criteria for the analysis. In our case, the criteria are the headings of each comparison chapter.
2. Generate weights for the criteria according their importance to the decision-makers. I have categorized the decision-makers into two categories: users and developers.
3. Evaluate the alternatives by calculating the sum of the rates multiplied by the weights. The alternatives here are IRAF and MIDAS. The rates are the summary scores at the end of each chapter. The rates have been normalized as out of 1.0.
4. The alternative with the highest summary score is the choice. Again, the summary score is normalized as out of 1.0, this means that summary scores of different decision-makers can be compared.

10.3 What is important to you?

Component (Attribute)	User Weights	Developer Weights
Software Architecture	1	2
Application Programming Environment	1	3
Graphical User Interface	2	1
System Requirements	1	1
System Installation	2	1
Real World Experiences	3	1
Current Status & Future Plan	2	3
Total weights	12	12

Table 10.1: Possible weights allocation to be used from a user and developer's perspective.

I have categorized the decision-makers into two categories: users and developers. As a result of their different requirements on the software system, each have differing weights for the scores. Table 10.1 shows my proposed weights for the users and the developers, which are briefly justified in the following paragraphs. Please note that any weighting scheme is heuristic and approximate, and never exact. Readers who disagree with my weights are welcome to substitute their own weights in the calculations which are provided in the later part of this chapter.

What are the features that a prospective user of an astronomical software package is looking for? Usually the astronomers just want to reduce and analyze their data. A little about its scripting language is all they are willing to learn. They are looking for user-friendliness, easy installation, a simple interface to the system, and an active development team which works on adding new functionality to the system. Therefore, I have assigned higher weights to these four criteria.

The developers' requirements are very different from the users'. They are looking for a well structured software architecture design, flexible programming environment and, most importantly, a sure future for the software

system which they are developing. Therefore, my suggested weights for these three criteria are higher than others.

In the next section, I will discuss the results from the weighted grading analysis. From the results of the analysis, I will conclude the section with my recommendations for both the users and the developers.

10.4 Which software is best for you?

By applying different weights to the criteria in the analysis, a definite change in the summary scores is clearly observable in Table 10.2. There is a big difference in the User-weighted scores for the two alternatives, while the Developer-weighted scores are almost the same. The following is my explanation of the summary scores.

10.4.1 Users

IRAF has been developed steadily by its programming group and other institutions. Instead of working on the area of user interfaces, the development has been focusing on providing more functionality. IRAF's GUI is supported with external utilities, which only work in 8-bit color display. In summary, IRAF is a feature-packed software that is difficult to install, and has an unintuitive user interface. A mediocre score of 0.59 is given by the analysis in Table 10.2.

For prospective users, MIDAS is clearly the better choice of the two. It scores a good 0.77 in the analysis. MIDAS meets almost all of the important criteria that I have set for the end-users in Table 10.1, except it has not been actively developed for the last six years. Since MIDAS already has a mature general image reduction and analysis facility, I believe it can satisfy most researchers. I wholeheartedly recommend MIDAS to those amateur astronomers who want a professional astronomical reduction and analysis software package that is simple to install and easy to use.

Component (Attribute)	Raw Scores		User-weighted Scores		Developer-weighted Scores	
	IRAF	MIDAS	IRAF	MIDAS	IRAF	MIDAS
Software Architecture	0.75	1.0	0.75	1.0	1.5	2.0
Application Programming Environment	0.67	1.0	0.67	1.0	2.01	3.0
Graphical User Interface	0.0	1.0	0.0	2.0	0.0	1.0
System Requirements	1.0	1.0	1.0	1.0	1.0	1.0
System Installation	0.6	1.0	1.2	2.0	0.6	1.0
Real World Experiences	0.5	0.75	1.5	2.25	0.5	0.75
Current Status & Future Plan	1.0	0.0	2.0	0.0	3.0	0.0
Total Score	4.52/7	5.75/7	7.12/12	9.25/12	8.61/12	8.75/12
Summary Score	0.65	**0.82**	0.59	**0.77**	0.72	**0.73**

Table 10.2: The Weighted Grading Analysis Results

10.4.2 Developers

From Table 10.2, we can see that IRAF and MIDAS score evenly with the weighted grading analysis for prospective developers. However, the summary scores do not tell the whole story because of the compensatory nature of the analysis.

The application developers for IRAF must use the SPP programming language. Nonetheless, with all the active developments around IRAF, the developers can be assured that their investment will not be in vain. A healthy score of 0.72 is given by the analysis in Table 10.2.

On the other hand, the developers for MIDAS have a choice of FORTRAN and C. Tools are available for these two programming languages to help the development of software. The down side of MIDAS is that it has not been actively developed for the last six years. Since MIDAS is available under the GNU General Public License (GPL), the source code is available to developers interested in providing new functionality. MIDAS scores 0.73 in the analysis.

For the developers, both software packages have their own advantages. However, these advantages are often offset by their disadvantages. It is my opinion that the developers should develop applications for the system that the users preferred. After all, the developers can adapt to the programming environment provided to them, and it is the users who are paying the developers' salaries!

10.4.3 Conclusion

IRAF is a feature-packed astronomical software that is difficult to install, and has an unintuitive user interface. MIDAS is a professional astronomical software that is simple to install and easy to use. I see no strong reason for the developer to choose one over the other. In my view a developer should develop applications for the system that local users prefer. To this end, this thesis recommends MIDAS to prospective users and developers.

10.5 Future Work

If I were given the opportunity, I would like to carry out several projects in the future:

Full-scale Real World Experiment - A full-scale real world experiment involving Astrophysics students from the Physics department could consolidate or disprove the conclusion of this thesis.

More Software Comparison - I would like to do an in-depth comparative study of other astronomical software packages on the market, including AIPS, AIPS++, IDL, CCDSoft, Astra Image and FTOOLS. Since this comparison will include both freeware and commercial software, it would help a prospective user to judge whether the features provided by the commercial software are worth extra dollars. The comparison would also cover the field of optical astronomy, radio astronomy, and space science.

Worldwide Survey - I believe a worldwide survey of astronomical software packages is required to improve the usability of the software. The target of the survey I envision is the professional and amateur astronomers around the world, who use computer programs to reduce and analyze their CCD images. The goal of this survey would be to provide user feedback to the developers, so that astronomical software in the future could be more user-oriented.

IDL - It would be interesting for someone to use IDL to implement a general CCD image reduction and analysis package. But the resulting program would require at least an IDL runtime license to run. The license is currently priced from USD $480 to $1080 [18]. In contrast, the major astronomical image reduction and analysis packages (IRAF, MIDAS, AIPS++) are freely distributed. IDL must offer significant and obvious advantages to overcome the price differential. One advantage of IDL is that it can run on the Windows 95/98/NT platform, while others programs cannot.

Appendix A

Interactive Data Analysis Environments

This section includes part of the comparison table [25] created in 1997 by Paul Barret (NASA) and Joe Harrington (MIT) for the FADS II (Future of Astronomical Data-analysis Systems) BOF Session at ADASS VI (Astronomical Data Analysis Software and Systems) on the topic of Interactive Data Analysis Environments [26].

The original table includes the following data analysis environments and scripting languages that are being used in a data analysis environment: Glish, IRAF, MIDAS, AVS, Khoros, Guile, IDL, Mathematica, Matlab, Octave, Phantom, PV-wave, Python, Rlab, S-plus, Scheme, Scilab, Smalltalk, Tcl, Java and Perl.

I have listed the software packages that are studied in this thesis, and three others (Mathematica, Java, Glish) for comparison purposes. Glish is used extensively by AIPS++, and is discussed in Section 9.3.4.

	IRAF	MIDAS	Glish	IDL	Mathematica	Java
GENERAL						
Sponsor	NOAO	MPE	ATBF	RSI	Wolfram	Sun
Version	2.x	2.0	0.9	5.0	3.0	1.0
Price(US$)				1400	1500	
Platform	Unix	Unix	Unix	Dos/Win Mac Unix	Dos/Win Mac Unix	Dos/Win Mac Unix
Source	yes	yes	yes	no	no	yes
Install						
type	binary source	binary source	binary source	binary	binary	binary source
method	makefile	makefile	makefile	program	program	makefile
compiler	optional	optional	optional	none	none	optional
Size						
core	60	30	5	20	30	10
library	?	?	300	?	?	?
Documentation	manual tutorial	manual tutorial	manual www	manual tutorial www	manual tutorial www books	manual tutorial www books
Support	email phone	email phone	email	email usenet phone	email usenet phone	email usenet
cost (US$/yr)				300	300	

Table A.1: Interactive Data Analysis Environments Table (Adapted from [25])

	IRAF	MIDAS	Glish	IDL	Mathematica	Java
ENVIRONMENT						
Cmd-line	custom	custom	custom	custom	custom	n/a
GUI	yes	yes	devel	yes	yes	yes
Extensible						
lang interface	no	no	C++	no	?	no
dyn linking	yes	yes	n/a	yes	?	yes
Debugger	no	?	?	no	?	no
Mem Manager	no	?	yes	no	?	?
Distributed	no	?	code	code	code	code
LANGUAGE						
Syntax like	C	fortran	fortran	fortran	fortran	C
Arg passing	value	?	ref/val	ref	value	?
Namespace	local	?	?	global	?	local
Classes	no	?	yes	no	no	yes
Modules	yes	?	yes	yes	yes	yes
Exceptions	no	?	yes	?	?	yes

Table A.2: Interactive Data Analysis Environments Table - continued

	IRAF	MIDAS	Glish	IDL	Mathematica	Java
NUMERICS						
Binary data	yes	yes	yes	yes	yes	yes
Symbolic	no	no	no	no	yes	no
Complex	no	?	no	yes	yes	no
Array						
indexing	fortran	?	fortran	fortran	?	C
slicing	yes	?	yes	yes	?	no
matrix ops	yes	?	element	yes	?	?
optimization	partial	?	partial	binary	?	?
LIBRARY						
Binary IO	?	?	?	yes	?	yes
CL options	?	?	?	?	?	?
Lists	no	?	no	no	?	yes
Regular expr	?	?	no	no	?	yes
String	yes	?	yes	yes	?	yes
System	yes	?	yes	yes	?	yes

Table A.3: Interactive Data Analysis Environments Table - continued

Appendix B

MIDAS Coding Rules

B.1 FORTRAN Coding Rules

To ensure portability for MIDAS application software, developers must follow coding rules. The FORTRAN coding rules are included here, quoted from "ESO-MIDAS Environment Document":

"Programs written in FORTRAN must conform to the ANSI FORTRAN-77 specifications defined by ISO Std. 1539-1980(E). The FORTRAN code must obey the following rules:

- All variables must be declared explicitly with type. This may be ensured by using the IMPLICIT NONE statement or using a compiler option which checks this.
- Indentation of at least two columns must be made for each level of IF/ELSE/ENDIF and DO/END/CONTINUE statements. For long DO-loops, additional comments at their start and end are strongly

encouraged. Further, statement labels must clearly indicate the loop structure and hierarchy.

- Statement labels should be given in increasing order and reflect the logical structure of the program. The number range 80000-81000 should be avoided because it is used by the MIDAS preprocessor.
- When using DO/CONTINUE statements, it is recommended to terminate the statement label after the DO with a ','(ref. ANSI FORTRAN-77) to avoid misinterpretation even through typing errors.
- Include files should not contain executable code statements.
- Each module must contain a comment header block which defines name, purpose, author, version etc.
- Subroutines and functions may share the source file with a main program only if they are not used in any other module.
- Internal READ statements using free or list-directed format are not allowed (e.g., READ(LINE,*)). The general usage of internal READ is deprecated due to usage of explicit editing formats.

"To make the code easier to understand, the usage of frequent comments, meaningful variable names and spaces between elements are recommended whereas GOTO statements are strongly discouraged." [14]

B.2 C Coding Rules

This section contains the C coding rules, quoted from "ESO-MIDAS Environment Document":

"Code written in the C language must conform to the ANSI definition of the Programming Language C as specified in ISO/IEC Std. 9899-1990. In addition, the following rules must be followed:

- Structures and unions must be defined by typedef declarations.

- Structures must always be passed to or returned by functions using pointers.
- Include files may not contain executable code statements.
- Each module must contain a comment header block which defines name, purpose, author, version etc.
- All modules using the interfaces defined in Appendix A must include the header file 'midas_def.h' which contains standard definitions and proto-types.
- Use of proto-types is encouraged however they should be made conditional on the definition of the '__STDC__' variable so that code still can be complied with K&R compilers.

"To make the code easier to understand, the usage of frequent comments, meaningful variable names and spaces between elements are recommended, whereas goto statements are strongly discouraged. The portability of the code must be verified by using the Unix lint utility or a similar tool. Any potential problems reported by this utility must be corrected." [14]

Bibliography

[1] 6th Southern Photometry Conference, June 1999. [Online]. Available: http://www.stardome.org.nz/PEP6.htm [2000, April 26].

[2] A. Accomazzi, C. Grant, et al. The World Wide Web and ADS services. In *American Astronomical Society Meeting*, volume 185, pages 4102+, December 1994.

[3] AIPS homepage, April 2000. [Online]. Available: http://www.cv.nrao.edu/aips/ [2000, April 26].

[4] AIPS++ homepage, April 2000. [Online]. Available: http://aips2.nrao.edu/docs/aips++.html [2000, April 26].

[5] Astrophysics Data System homepage, January 2000. [Online]. Available: http://adsabs.harvard.edu/ [2000, April 26].

[6] P. Ballester and K. Banse. GUIs in the ESO-MIDAS environment. In *ASP Conf. Ser. 52: Astronomical Data Analysis Software and Systems II*, volume 2, pages 357+, 1993.

[7] C. Bamrick (cbamrick@rsinc.com). Private communication, January 2000. E-mail to Ching Yi Tsai (tsai@ihug.co.nz).

[8] G. Bothun. CCDs for material scientists, (No date). [Online]. Available: http://zebu.uoregon.edu/ccd.html [2000, April 26].

[9] A. Bridle and E. Greisen. The NRAO AIPS project - A summary. In *AIPS Memo 87*. NRAO, April 1994.

[10] T. Cornwell (tcornwel@cv3.cv.nrao.edu). Private communication, January 2000. E-mail to Ching Yi Tsai (tsai@ihug.co.nz).

[11] Cottrell and Mackie. New Zealand astronomy and astrophysics in the first decade of the new millennium, December 1998. [Online]. Available: http://www.vuw.ac.nz/~mackie/FP/Astron_Astrophys_text.html [2000, April 26].

[12] L. Davis and D. Tody. IRAF newsletter - Number 14, April 1998. [Online]. Available: http://iraf.noao.edu/iraf/web/irafnews/apr98/irafnews.11.html [2000, April 26].

[13] ESO. *Installation of MIDAS on UNIX Systems*, July 1998.

[14] ESO-MIDAS environment document, June 1995. [Online]. Available: http://www.eso.org/projects/esomidas/doc/env/ [2000, April 26].

[15] ESO-MIDAS homepage, January 2000. [Online]. Available: http://www.eso.org/projects/esomidas/ [2000, April 26].

[16] ESO-MIDAS requirements, October 1998. [Online]. Available: http://www.eso.org/projects/esomidas/midas-require.html [2000, April 26].

[17] ESO-MIDAS system overview, October 1998. [Online]. Available: http://www.eso.org/projects/esomidas/midas-overview.html [2000, April 26].

[18] K. Falco (KFalco@rsinc.com). Private communication, March 2000. E-mail to Ching Yi Tsai (tsai@mac.com).

[19] M. J. Fitzpatrick. SPPTOOLS: Programming tools for the IRAF SPP language. In *ASP Conf. Ser. 52: Astronomical Data Analysis Software and Systems II*, volume 2, pages 213+, 1993.

[20] M. J. Fitzpatrick (fitz@noao.edu). Private communication, May-December 1999. E-mail to Ching Yi Tsai (tsai@ihug.co.nz).

[21] FY2000 budget overview, March 1999. [Online]. Available: http://www.research.umich.edu/research/news/research_reporter/fy99/990315/Summary.html [2000, April 26].

[22] B. E. Glendenning. Creating an object-oriented software system – The AIPS++ experience. In *ASP Conf. Ser. 101: Astronomical Data Analysis Software and Systems V*, volume 5, pages 271+, 1996.

[23] P. Grosbol. MIDAS. In *Reviews of Modern Astronomy*, volume 2, pages 242–247, 1989.

[24] M. Hadfield (m.hadfield@niwa.cri.nz). Private communication, November-January 2000. E-mail to Ching Yi Tsai (tsai@ihug.co.nz).

[25] J. Harrington and P. E. Barrett. Interactive data analysis environments, 1997. [Online]. Available: http://lheawww.gsfc.nasa.gov/users/barrett/IDAE/table.1.html [2000, April 26].

[26] J. Harrington and P. E. Barrett. Interactive data analysis environments BoF session. In *ASP Conf. Ser. 125: Astronomical Data Analysis Software and Systems VI*, volume 6, pages 69+, 1997.

[27] H. Haubold. Workshop on data analysis. In A. Batten, editor, *IAU Commission 46 Online Newsletter*. IAU, 1998. [Online]. Available: http://yan.open.ac.uk/IAU46/newsletter48.html [2000, April 26].

[28] B. Hutton. Private communication, March 2000.

[29] IDL frequently asked questions, (No date). [Online]. Available: http://www.ivsoftware.com:8000/FAQ/ [2000, April 26].

[30] ION, IDL on the Net homepage, 2000. [Online]. Available: http://www.rsinc.com/ion/index.cfm [2000, April 28].

[31] IRAF frequently asked questions, March 1999. [Online]. Available: http://iraf.noao.edu/faq/FAQ.html [2000, April 26].

[32] IRAF homepage, April 2000. [Online]. Available: http://iraf.noao.edu/ [2000, April 26].

[33] IRAF installation documentation, February 2000. [Online]. Available: http://iraf.tuc.noao.edu/docs/igsm.html [2000, April 26].

[34] T. Jenness, F. Economou, et al. Perl at the joint astronomy centre. In *ASP Conf. Ser 172: Astronomical Data Analysis Software and Systems VIII*, volume 8, pages 494+, 1999.

[35] Ousterhout J. K. Scripting: Higher-level programming for the 21st century. *Computer*, 31(3):23–30, March 1998.

[36] B. Landon. Making selection decisions - Some background. [Online]. Available: http://www.ctt.bc.ca/landonline/whitpapr.html [2000, April 26].

[37] LessTif homepage, April 2000. [Online]. Available: http://www.lesstif.org [2000, April 26].

[38] Z. G. Levay. *SPP Reference Manual*. STSDAS Group, Science Software Branch, 2nd edition, October 1992.

[39] G. Longo and G. Sedmak. Acquisition, processing and archiving of astronomical images. In Giorgio Longo, Guiseppe; Sedmak, editor, *Seminar held in Anacapri, 1989, Capodimonte: Osservatorio Astronomico, 1990*, 1990.

[40] E. Mandel and S. S. Murray. Other people's software. In *ASP Conf. Ser. 145: Astronomical Data Analysis Software and Systems VII*, volume 7, pages 142+, 1998.

[41] P. Martinez and A. Klotz. *A practical guide to CCD astronomy*, volume 8 of *Practical Astronomy Handbooks Series*. Cambridge University Press, 1998.

[42] MOTIF 2.0 technical overview, January 1999. [Online]. Available: http://www.opengroup.org/tech/desktop/motif/xjournal.htm [2000, April 26].

[43] P. Nass (pnass@eso.org). Private communication, January–April 2000. E-mail to Ching Yi Tsai (tsai@mac.com).

[44] Obtaining AIPS++, (2000?). [Online]. Available: htpp://aips2.nrao.edu/docs/reference/install.html [2000, April 26].

[45] OSF/MOTIF homepage, March 2000. [Online]. Available: http://www.opengroup.org/tech/desktop/motif/ [2000, April 26].

[46] V. Paxson. Glish: A software bus for high-level control. In *Proceedings of the International Conference on Accelerator and Large Experimental Physics Control Systems*, 1993.

[47] G. S. Pennycook. Modification of the Boller and Chivens telescope to f/6.25 and photometry of IC 5249's halo. Master's thesis, Physics Department, University of Auckland, 1998.

[48] M. R. Pérez. Alternative astronomical software packages. *Astrophysics and Space Science*, 258:271+, 1998.

[49] N. Pirzkal and R. N. Hook. Python in astronomy. In *ASP Conf. Ser. 172: Astronomical Data Analysis Software and Systems VIII*, volume 8, pages 479+, 1999.

[50] J. Preece. *Human-Computer Interaction*, chapter 1, pages 17–18. Addison-Wesley Pub. Co., 1994.

[51] R. Pressman. *Software Engineering - A Practitioner's Approach.* McGraw-Hill, Inc, third edition, 1992.

[52] Requirements for data analysis software for the Green Bank Telescope, May 1991. [Online]. Available: http://www.gb.nrao.edu/GBT/memos/memo72.txt [2000, April 26].

[53] Research Systems, Inc. homepage, (No date). [Online]. Available: http://www.rsinc.com [2000, April 26].

[54] D. R. Schiebel. Programming in Glish. In *ASP Conf. Ser. 101: Astronomical Data Analysis Software and Systems V*, volume 5, pages 315+, 1996.

[55] E. Sturm. IDL in astronomy: ISO and the ISAP project. *Astrophysics and Space Science*, 258:285–292, 1997.

[56] H. Titus and A. Waks. *Image Acquisition and Scientific Imaging Systems.* Spie, 1994.

[57] D. Tody. The IRAF data reduction and analysis system. *Photo-Optical Instrumentation Engineers*, 627:733+, 1986.

[58] D. Tody. IRAF in the nineties. In *ASP Conf. Ser. 52: Astronomical Data Analysis Software and Systems II*, volume 2, pages 173+, 1993.

[59] D. Tody. A portable GUI development system – The IRAF widget server. In *ASP Conf. Ser. 77: Astronomical Data Analysis Software and Systems IV*, volume 4, pages 89+, 1995.

[60] D. Tody. Message bus and distributed object technology. In *ASP Conf. Ser. 145: Astronomical Data Analysis Software and Systems VII*, volume 7, pages 146+, 1998.

[61] D. Tody and M. J. Fitzpatrick. PC-IRAF: The choice of a GNU generation. In *ASP Conf. Ser. 101: Astronomical Data Analysis Software and Systems V*, volume 5, pages 322+, 1996.

[62] D. Tody (tody@noao.edu). Private communication, December 1999. E-mail to Ching Yi Tsai (tsai@ihug.co.nz).

[63] UNIX/IRAF site manager's guide, December 1997. [Online]. Available: http://iraf.noao.edu/iraf/web/docs/src/pcix/unixsmg-toc.html [2000, April 26].

[64] G. van Moorsel, A. Kemball, and E. Greisen. AIPS developments in the nineties. In *ASP Conf. Ser. 101: Astronomical Data Analysis Software and Systems V*, volume 5, pages 37+, 1996.

[65] VLT Data Flow System homepage, October 1999. [Online]. Available: http://www.eso.org/projects/edfs/ [2000, April 26].

[66] VLT software development, January 1999. [Online]. Available: http://www.eso.org/projects/vlt/sw-dev/ [2000, April 26].

[67] S. Wampler. Off-the-shelf control of data analysis software. In *ASP Conf. Ser. 61: Astronomical Data Analysis Software and Systems III*, volume 3, pages 511+, 1994.

[68] K. Weatherall (kweather@cv3.cv.nrao.edu). Private communication, January 2000. E-mail to Ching Yi Tsai (tsai@ihug.co.nz).

[69] W. Young and D. A. Roberts. Demonstration of parallel AIPS++ deconvolution application. In *ASP Conf. Ser. 172: Astronomical Data Analysis Software and Systems VIII*, volume 8, pages 506+, 1999.

Name Index

List of Acronyms

ADASS: Astronomical Data Analysis Software & Systems

ADS: Astrophysics Data System

AIPS: Astronomical Image Processing System

ALMA: Atacama Large Millimeter Array

APL: A Programming Language

BOF: Birds Of a Feather

CCD: Charged-Coupled Device

CDE: Common Desktop Environment

CL: Command Language

DCL: Digital Command Language

DFS: Data Flow System

ESA: European Space Agency

ESO: European Southern Observatory

EUVE: Extreme UltraViolet Explorer

FADS: Future of Astronomical Data-analysis Systems

FAQ: Frequently Asked Questions

FIFO: First In First Out

FITS: Flexible Image Transport System

GBT: Green Bank Telescope

GPL: General Public License

GUI: Graphical User Interface

HSI: Host System interface

HST: Hubble Space Telescope

IDL: Interactive Data Language

IDLDE: IDL Development Environment

IHAP: Image Handling And Processing system

IRAF: Image Reduction and Analysis Facility

ISAP: ISO Spectral Analysis Package

ISO: Infrared Space Observatory

IUE: International Ultraviolet Explorer

LASP: Laboratory for Atmospheric and Space Physics

LBL: Lawrence Berkeley Laboratory

MCDA: Multiple Criteria Decision Analysis

MCL: MIDAS Command Language

MIDAS: Munich Image Data Analysis System

MMED: Mars Mariner Spectrum Editor

MOA: Microlensing Observations in Astrophysics

MPI: Message Passing Interface

NCSA: National Center for Supercomputing Applications

NOAO: National Optical Astronomy Observatories

NRAO: National Radio Astronomy Observatory

NSF: National Science Foundation

OECD: Organization for Economic Co-operation and Development

OOP: Object-Oriented Programming

PDF: Portable Document Format

ROSAT: ROentgen SATellite

RSI: Research Systems, Inc.

SDAS: Science Data Analysis System

SOHO: SOlar and Heliospheric Observatory

SPP: Subset Pre-Processor

STScI: Space Telescope Science Institute

STSDAS: Space Telescope Science Data Analysis System

UIDS: User Interface Design Systems

VLA: Very Large Array

VLBA: Very Long baseline Array

VLBI: Very Long Baseline Interferometry

VLT: Very Large Telescope

VOS: Virtual Operating System

WYSIWYG: What You See Is What You Get