IRAF in the Nineties

Doug Tody[1]

National Optical Astronomy Observatories, Tucson, AZ 85726

Abstract. The IRAF system (Image Reduction and Analysis Facility) has been under development since 1981 and in use since 1984. Currently, in 1992, IRAF is a mature system with hundreds of applications which is in wide use within the astronomical community. After a brief look at the current state of IRAF, this paper focuses on how the IRAF system is expected to develop during the coming decade. Certain key new technologies or trends which any new data analysis system will need to deal with to be viable in the nineties and beyond are discussed. An overview of the planned enhancements to the IRAF system software is presented, including work in the areas of image data structures, database facilities, networking and distributed applications, display interfaces, and user interfaces.

1. Introduction

The IRAF data reduction and analysis system has been around since 1981. Today IRAF is a mature system with hundreds of applications which is supported on all major platforms. Many institutions, projects, and individuals around the U.S. and the world have developed software for IRAF. Some of these packages are comparable in size to the IRAF core system itself.

At the present time there are half a dozen large groups developing software for IRAF, plus many individuals or small groups. Coordination of the work being done by the large groups is the responsibility of the IRAF TWG (Technical Working Group), an interagency group which oversees IRAF software development. Scientific review of IRAF development is provided by an IRAF User's Committee, which oversees IRAF as a whole and which reports to NOAO, by additional User's Committees reporting on the various projects developing large layered packages for IRAF, and by the staff and management of the institutions funding IRAF development. IRAF is used primarily by the ground based astronomy (NSF) and NASA space astrophysics communities.

A list of the IRAF layered packages currently installed at NOAO/Tucson is shown in Figure 1, to illustrate the variety of packages available. This list is not all-inclusive, i.e., there are additional layered packages available for IRAF other

[1] NOAO is operated by AURA, Inc. under contract to the National Science Foundation.

than those shown here, which were the ones which happened to be installed at NOAO when this figure was prepared. The standard IRAF distribution itself, consisting of the core IRAF and NOAO package trees, contains about 50 additional packages, or several hundred tasks, totaling approximately 1.3 million source lines. The core IRAF system includes the IRAF system software (host system interface, run time and programming environments, command language and other user interfaces, and core applications) and is required to compile and run any layered software.

adccdrom	tools for accessing ADC CD-ROM
ccaccq	IRAF CCD data acquisition
color	prototype RGB rendering tasks
ctio	CTIO local tasks
demos	IRAF demos
ftools	FITS tools package
grasp	GONG data processing (helioseismology)
ice	IRAF CCD data acquisition
iue	tools for importing IUE spectral data into IRAF
mem0	maximum entropy image restoration
nlocal	NOAO/Tucson local tasks
nso	Solar astronomy
spptools	SPP programming utilities
steward	Steward observatory local tasks
stsdas	STScI (HST) data processing
tables	STScI table tools package
vol	volume rendering
xray	SAO x-ray data analysis package

Figure 1. IRAF layered packages installed at NOAO (Dec. 1992)

As of late 1992 the current release of IRAF, which is still in distribution, is version 2.10. As of December 1992 there were a total of 1068 logged distributions of the previous version of IRAF, V2.9, of which 196 distributions were tape distributions mailed to the user at cost, and 872 distributions were downloaded via anonymous ftp from the IRAF network archive on iraf.noao.edu. An unknown number of additional distributions were downloaded via DECNET network transfer (we don't know how many, as we log only ftp file transfers). These statistics count only distributions leaving NOAO; since the system is freely available, we have no way of recording redistribution of the system at remote sites or within large institutions. IRAF site support traffic totals over 5000 email messages or phone calls per year, counting both incoming and outgoing messages. Based on the number of distributions and the site support traffic we estimate there are currently several thousand active users of IRAF.

The remainder of this paper focuses on where IRAF is headed over the remainder of this decade. We look first at some key new technologies that we feel IRAF (or any modern astronomical data analysis system) must use effectively to be competitive by the end of the decade. Some pitfalls that we feel system developers would be wise to consider are also discussed. Finally, we summarize the work planned for the next few years to enhance the IRAF system software to meet these new challenges.

2. Key new technologies for the next decade

IRAF is a long term project. Most of the software or hardware technologies upon which IRAF is based typically have a lifetime of only 5 or 10 years - considerably less than the expected lifetime of the IRAF software. To remain up to date it is necessary to make use of new technology, but one must do so carefully to avoid tying the software irrevocably to a technology which will one day become obsolete. It is not easy to change a large system once a direction has been chosen, so one must take the long term view, always planning 5 to 10 years in the future, trying to visualize what future computer systems will be like and what we want our software to look like on those systems.

2.1. Key new technologies

The following are some key new technologies and technological issues or trends which we feel any modern data analysis system should be concerned with.

User interfaces

As computer systems become more powerful, software systems are becoming larger and more complex. People do a lot more with computers now than they did a few years ago. Functionality and efficiency, while still important, are no longer the overriding concerns they once were. The issues of managing complexity, and ease of use, are increasingly important concerns. The challenge of user interface design is to make complex systems comprehensible, intuitive, and easier to learn and use. Sophisticated user interfaces will make our software more pleasant to use, and allow more complex and sophisticated applications to be written. The days are past when the user interface can be taken for granted when designing new software.

High level languages

Our common everyday computers are becoming so powerful that most of the compute cycles now go to waste. At the level of compiled code, our computer languages and software systems are becoming increasingly complex, to the point where it may take an expert with years of training to deal with them. It may be that the time is rapidly passing when casual users will do very much programming with general purpose compiled languages like Fortran, C, C++, and so on. The trend is towards higher level interpreted languages which are tailored for a particular type of application. Whether these languages are syntax driven, visual, or whatever does not really matter; in general the optimum type of language depends upon how the language will be used, and languages should be customized for a particular application. In the future, users will still develop custom applications, but they will increasingly do so using sophisticated, high level, application specific custom languages which are embedded in feature-rich data processing environments.

Networking and distributed objects and data

Our computers are getting powerful enough that for many applications, further gains in compute power won't make a whole lot of difference. A powerful computer and sophisticated software aren't worth much unless one has data or other raw information of some type to process, analyze, or query. Fortunately a new way has been found to expand the capability of a computer system: the growth of global networks is opening up a whole new dimension on what we can do with computers. It is already the case that one can do wondrous things with even the simplest hand held computer - so long as it is connected to the global networks. The networks give us access to an inconceivable amount of data or information of various types. Not only do the networks provide access to vast amounts of raw information, they make it possible to export arbitrary *services* via the network. Rather than export data or software, one can now export services which remote clients can access at runtime to do any number of interesting things. We are only just starting to learn how to make use of the global networks, but it is already clear that the networks will change forever how we do computing, and how we use computers.

Object oriented software structure and methodology

Every few years something new comes along (e.g., AI, CASE, HyperCard, etc.) which proponents claim will revolutionize how we do computing. Most of these "silver bullets", useful though they may be in some applications, are oversold and after a time something else gets all the press. The latest such hot item is object oriented programming (OOP). The journals are full of talk about object oriented languages, databases, programming tools, and so on. This time it is not just another overhyped product though. The object oriented approach to software development and software design is probably the most important development in software engineering since structured programming in the 80's. In fact it is a natural outgrowth of the best software practices of the 80's.

What is important about object orientation is not a particular language, commercial product, or other tool, but the concepts and methodology underlying the object oriented approach to software development and systems design. In particular, the object oriented approach places a special emphasis on the *conceptual modeling* of the objects (classes) comprising a software system. This emphasis on conceptual modeling, and the encapsulation or information hiding that is a natural part of the object oriented approach, are fundamentally important in dealing with the complexity of large modern software systems. Other elements of the object oriented approach such as subclassing and inheritance are probably fundamental to a true object oriented software structure, but this is a fairly specific software structure, and not necessarily the best one for all applications. Like any technology, OOP will be better for some things than for others, and we are still learning the limitations of this new technology and where it can be used most effectively.

Database technology

There is nothing very new about database technology. To date though, astronomy has done little with database technology, beyond its obvious use for indexing data archives. We think that there is much that could be done by combining, e.g., database technology with a graphical user interface to perform sophisticated queries of the catalogs produced by astronomical analysis programs such as are produced by image classification, source detection, and stellar or galaxy photometry programs. Furthermore our data sets are becoming larger and increasingly more complex, as is the way we access data, especially when we take network access to remote databases into account. Database technology will eventually have to be brought into play to effectively manage this increased complexity. Conventional relational database technology will continue to be important for large data archives and for some types of catalogs produced by analysis programs, but object oriented database techniques will be better suited to the complex data objects dealt with by our online analysis systems.

2.2. Concerns

In the process of employing all this new technology there are a number of things to watch out for.

The coming OS wars

UNIX is king right now, but will this be the case ten years from now? Ten years ago the dominant system in astronomy was the VAX running VMS. Today it is the UNIX workstation. UNIX is a very good system and it (or more properly its descendents) might still be the dominant system ten years from now, but this is by no means certain. There is a very real possibility that the dominant system for astronomers ten years from now could be the PC. Not the PC we have now, but the PC we will have then, when a PC is more powerful than the workstation of today, cheaper, portable, fully connected to the networks, and capable of running "shrink wrapped" personal applications in addition to specialized astronomical software. The engineering workstations and servers of today will still be around, and they will be more powerful than ever, but an increasing share of scientific computing is likely to be done on mass market PC systems.

If the PC does so well then we cannot be certain that UNIX will still be the predominant operating system ten years from now. We might instead be using an operating system which was designed for the mass market, such as Windows/NT, or conceivably even some future version of MacOS (most likely a mixture of all these). UNIX may be more powerful, more elegant, more technically superior, and less proprietary, but those criteria will not necessarily prove as compelling to the mass markets as they have to the academic and engineering markets. Even within the UNIX community there is still considerable variation in what we call UNIX, and despite efforts like POSIX there is no real evidence that this situation will ever change. On the contrary, UNIX systems are becoming increasingly complex and small differences are correspondingly magnified.

Window systems

In the past the main concern when porting software to a new platform was the operating system. Operating system differences are still a concern, but not as big a concern as in times past. A possibly more significant problem, and one which is perhaps being overlooked by many folks now writing software, is the window system, or in the case of X, the window system toolkit. Modern window systems are comparable in complexity to operating systems but the technology is much newer, and is still evolving rapidly. It is likely that any window system specific software written today will have to be thrown out and rewritten a few years from now. Most window system specific software today would have to be largely rewritten to "port" the application to a different window system on a different platform. Despite these problems, most window system applications written today are monolithic applications with the application specific functions and user interface code tightly interwoven. Window systems may prove to be the "assembly language" of the 90's.

Computer languages

In the past ten years the major players in the general computer languages arena have changed, but the game has not. We have seen Fortran 77, K&R C, ANSI C, Fortran 90, and lately C++, with others such as Ada, Pascal, and Objective C on the sidelines. Computer languages are constantly evolving. Even given language standards, implementations of a language by vendors on different hardware vary considerably. This is unlikely to ever change.

The evolution of computer languages is not a problem so long as one is content to write disposable software. If the projected lifetime of a body of software is ten years or longer, and the body of software is large enough that rewriting it may not be practical, the evolution of languages is a serious problem which may eventually cause the software to become obsolete, along with the technology it has been tied to. Even in the short run, the variation in language implementations on different platforms can be a serious support headache if the body of code is sufficiently large.

3. Major IRAF system software enhancements

In this section we describe the work being done to enhance the IRAF system software. This is a long term effort extending over a period of years. Some of the work discussed has already been completed, but much of it is either in progress or still to be done.

The work presented here attempts to exploit the new technologies discussed in the last section, while avoiding the pitfalls that can come from tying software too closely to a particular technology. Existing technology is only useful up to a point; much of the work discussed in this section is an outgrowth of the work already done on the IRAF system, and reflects problems some of which appear to be unique (at some level) to astronomical data analysis.

The reader is assumed to already have some familiarity with the current IRAF system software. The software described here is very extensive and it is impossible in a short review article like this to go into very much detail, or explain all the terminology used.

3.1. Image Structures

The term "image structures" refers to the representation of the primary data type in IRAF, the *image*. In IRAF an image is not a simple picture, but an arbitrarily complex data object. An image consists of an N-dimensional logical data raster (sampled data array) plus various bits of associated information, some of which may be quite complex objects in their own right. The logical data raster need not be physically stored as a sampled data array, for example in the case of event data the data is stored as an event list and sampled only when the image is accessed. The information often associated with a data raster includes history information, any attributes computed by analysis of the image data, world coordinate systems, pixel or region masks, uncertainty or "noise" information, and so on.

A common example of an image is a raw data image, i.e., an astronomical observation. Examples of raw data images are a 1D spectrum, a 2D CCD data frame, or a 3D Fabry-Perot or radio spectral image cube. Images of dimension higher than 3 are rare in astronomy. Typical astronomical data sets can be quite large, e.g. several gigabytes, perhaps consisting of thousands of small spectra, or several hundred large 2D images. Individual images of 32 megabytes or larger are occasionally seen.

high level image class	
`IMIO`	image i/o
image header access	
`IMIO`	header access
`DFIO`	datafile manager (*)
`FMIO`	file manager
image kernels (internal to IMIO)	
`IKI`	image kernel interface
`OIF`	old (original) image format
`STF`	HST image format
`PLF`	pixel list image format
`QPF`	QPOE (event list) image format
`FTF`	FITS image format
new format	new DFIO based image format (*)
others	HDS(?), "PC" image formats (*)
auxiliary classes	
`MWCS`	world coordinate systems
`PMIO, PLIO, MIO`	pixel masks or lists
`QPOE`	event list data files
`NFIO`	noise function package (*)

Figure 2. New Image Structures

When IRAF was first released some years ago the only image format consisted of a pair of files per image, one for the header and one for the pixel matrix, with the header file consisting of a fixed binary structure plus a variable number of FITS cards (not a terribly flexible or efficient structure). Over time several alternative image formats have been added, as well as support for some of the auxiliary data objects associated with images. It has become increasingly difficult to store all this information in the simple data structures provided by the older IRAF image formats. The purpose of the new image structures project is to provide a general, well integrated hierarchy of image object classes for flexibly and efficiently representing a wide variety of image data.

The major components of the new image structures are summarized in Figure 2. The new image structures project is further along than most of the other new software discussed in this paper; everything listed has been implemented except for the items marked with an asterisk. This is a *big* project; some of the subsystems listed here, e.g., MWCS, QPOE etc., are major projects in their own right, and in total the new image structures code will likely exceed 100K lines, not counting the lower level IRAF classes or other library code.

A key feature of the implementation of the image interface in IRAF is the *image kernel*. An image kernel is the only part of IRAF that knows anything about how an image is stored externally, i.e., the physical image format. The image kernel implements a mapping between the physical image format and the logical view of an image implemented in the runtime image descriptor used to access an active image object. The image kernel can provide a standard interface to a wide variety of image types, including odd things like photon event lists, image masks, or image display server frame buffers, in addition to various standard image raster disk file formats. In principle IRAF can be integrated with any external image processing system by implementing an IKI image kernel for the image format defined by the external system. The best example of this is FITS. The FITS image kernel also allows archival data, e.g. on CD-ROM or on a remote network server, to be directly accessed by IRAF programs.

The main work remaining to be done to finish the new image structures project is to implement a new standard IRAF online image format based on the general datafile manager (DFIO, discussed in the next section). This will replace the existing OIF image format. The new format will make it possible to simply and efficiently group complex objects such as world coordinate systems, pixel masks, and compressed pixel uncertainty arrays with pixel arrays to form the objects we call images.

3.2. Database facilities

Managing complex data structures such as the image structures in a flexible and efficient manner, while providing features such as data independence, machine independence, a data recovery capability, transparent storage of arbitrarily large data elements, capabilities for storing complex objects (not just simple tables), indexing for efficient lookup, and a good integration with the higher level IRAF software, is a complex and demanding problem. The low level interface planned to provide this capability for IRAF is DFIO, the data file manager. DFIO is layered upon FMIO, the file manager, which is in turn layered upon IRAF binary file i/o (FIO). DFIO is a medium level interface designed for embedded applications, e.g. it will be used internally within IMIO to store image data. For the most part IRAF applications will not use DFIO directly, rather they will deal with data at a higher level, e.g. via the image class.

One of the types of data DFIO will be capable of storing is the table, as in a relational database. Hence, in addition to its use as an embedded interface within IRAF system software to store complex data objects, DFIO will provide a traditional relational database capability for applications such as catalog access. Since IRAF already provides a builtin networking capability, DFIO will automatically be usable in client-server applications to provide a distributed database facility. To be able to access external, non-IRAF databases, a database server architecture will be used (similar to the use of image kernels by IMIO).

3.3. Networking and distributed applications

Networking is an integral part of IRAF and IRAF has always been able to support distributed applications. In IRAF all access to external resources is via the IRAF kernel. The IRAF kernel has a builtin remote procedure call facility allowing kernel procedures to execute either locally or remotely. (This includes *all* kernel procedures that access a named external resource, be it a file, directory, tape drive, image display, process, or whatever). In a local reference the procedure is executed directly; when the resource being accessed resides on a remote node a custom RPC protocol layered upon the IRAF networking driver is used to remotely execute the kernel procedure. The only system dependent part of all this is the networking driver, which can use any standard message or stream oriented transport layer, e.g. TCP/IP, DECNET, and so on. Since the IRAF kernel provides a standard host interface, the routing or leaf nodes in a distributed IRAF process tree can execute on host machines running any operating system to which IRAF has been ported. It is even possible to transparently route RPC calls between different networks, e.g. the Internet and SPAN.

There are still some significant enhancements planned for the IRAF networking system but these are for the most part comparatively minor evolutionary enhancements. One of the most interesting enhancements being considered is some sort of interface to non-IRAF servers, e.g. ftp or WAIS servers. This would not provide the full capability of the IRAF kernel, but might work for simple directory and file access, and would allow any IRAF application to transparently access arbitrary servers on the network whether or not they provide an IRAF kernel server.

3.4. User Interfaces

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. This directly reflects the priorities for IRAF development in the late 1980's, which emphasized getting numbers out of the data. This meant new applications, and due to the common environment, enhanced system support for these applications (e.g. the new image structures). In the early 1990's, with a wealth of software now in the system and more people than ever using IRAF, the emphasis has shifted towards ease of use and improved user interaction and data display.

Enhancements to the IRAF user interfaces are planned in many areas. Two of the most exciting are a general GUI (graphics user interface) capability, available to any IRAF application and capable of making full use of the advanced capabilities of modern window systems, and less obviously, something called minilanguage support.

Modern window systems are remarkably complex software systems, and the field is still evolving rapidly, with many quite different window systems and

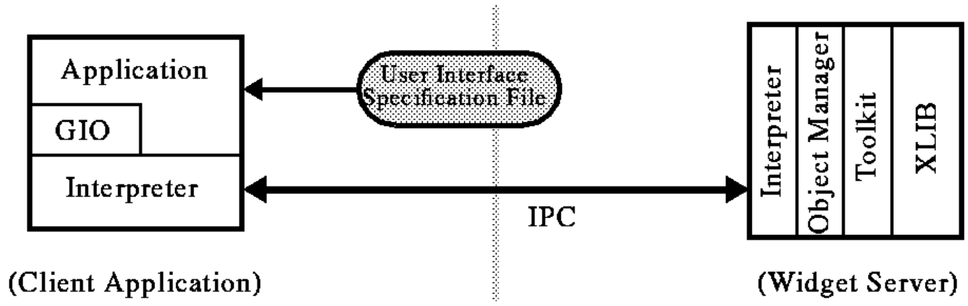

Figure 3. Widget Server Architecture

window system toolkits being developed or in use. Using window user interfaces effectively in scientific applications is challenging, as if one is not careful and the wrong approach is taken, programmers may spend all their time struggling with complex window system software and not get any science software written. Due to the complexity of the field, learning a particular toolkit or user interface builder is time consuming and a considerable investment in time is required to make use of any particular tool. A wrong decision could result in a great deal of wasted time, particularly for a large project where many people may be developing software.

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.

This architecture has many advantages, e.g., a complete separation of the user interface and functional code, and a high level interpreted interface to the window system for the programmer, making it easy to develop GUIs (and easy for the user to customize the user interface). Since only the widget server knows about a particular window system or toolkit, the widget server also provides window system and toolkit independence, allowing a new window system or toolkit to be supported merely by implementing a new version of the widget server. The widget set provided by the widget server will include the standard toolkit text, button, scollbar, list, geometry, etc. widgets, plus some custom widgets (such as graphics and image display widgets) tailored to IRAF applications.

As powerful as the graphics user interface can be, it is not the only way to do a user interface, nor is it necessarily the best type of user interface for all interactive applications. A quite different type of applications user interface which is at least as powerful, and also well suited to complex applications, is the context-based minilanguage. A minilanguage is a single program (IRAF task) with a syntax driven, command line user interface. The program maintains an internal state and successive input statements modify this state. The syntax, command or function set, and internal data structures are customized for each application. The individual functions are usually simple, but arbitrarily complex operations can be performed by stringing together sequences of commands or expressions. By designing an appropriate syntax very powerful applications-specific languages can be devised.

A good language will be extensible, allowing users to define new procedures, link to external compiled routines, or interface external IRAF or host tasks so that they appear as functions in the minilanguage. It will even be possible to combine a graphics user interface with a minilanguage. For example, the combination of the widget server with a minilanguage will provide both a fully featured GUI capability and a powerful interpreted computing engine, both programmable by the user without need to resort to low level compiled languages. When all this is layered upon the IRAF environment, providing well integrated access to powerful facilities such as the IRAF image structures and a wealth of existing external tasks, the result will be high level applications of unprecedented power, flexibility, and sophistication.

Acknowledgments. Without the contributions of many people over the years, the IRAF system we have now would not exist. The author particularly wishes to thank Frank Valdes and Lindsey Davis of the NOAO IRAF group, who wrote much of the IRAF software. STScI and SAO have made major contributions to the system over the years and the IRAF project would not be the same without their involvement. A grant from the NASA astrophysics data program has made all the difference as IRAF use continued to grow while the NOAO budget continued to shrink. Finally, we wish to thank the NOAO directors and scientific staff for their continuing support and enthusiastic use of IRAF, and for their help in making IRAF a better system.

References

Tody, D., 1986, "The IRAF Data Reduction and Analysis System", in *Instrumentation in Astronomy VI*, David L. Crawford, Editor.

Hanisch, R. J. 1991, "STSDAS: The Space Telescope Science Data Analysis System", in *Data Analysis in Astronomy IV* (Di Gesù, V., Scarsi, L., Buccheri, R., Crane, P., Maccarrone, M.C., and Zimmermann, H.U., eds., Plenum Press, New York), 97.

Worrall, D.M., Conroy, M., DePonte, J., Harnden, F.R., Mandel, E., Murray, S.S, Trinchieri, G. , VanHilst, M. , Wilkes, B.J., 1992, "PROS: Data Analysis for ROSAT" in *Data Analysis in Astronomy IV*, eds. V. Di Gesu *et al.*, Plenum Press, 145.

Olson, E. C. and Christian, C. A., 1992, "The EUVE Guest Observer Analysis Software." in *Astronomical Data Analysis Software and Systems I.*

See also the many technical papers describing the IRAF software, available in `iraf.noao.edu:iraf/docs`.