D. RECENT SOFTWARE DEVELOPMENTS

https://doi.org/10.1017/S0252921100082737 Published online by Cambridge University Press

SYSTEM SOFTWARE APPROACHES TO THE ANALYSIS OF MULTIDIMENSIONAL DATA STRUCTURES

Rudolf Albrecht
Institute for Astronomy, University of Vienna
Space Telescope Science Institute, Baltimore

ABSTRACT

In this review paper on software techniques, the concepts of "software system" and "data base interface" are defined in the context of astronomical data analysis requirements. Principles of software design and maintainance over are discussed.

A deliberate effort was made to stay as close as possible to the astronomical application and not deviate too much into computer science.

1. INTRODUCTION

The analysis of multidimensional data structure is more commonly known as Image Processing. Image processing is of growing importance for modern astronomical data analysis. At the Space Telescope Science Institute, a data analysis system for data generated by the Science Instruments of the Space Telescope is being defined. Although I will in this presentation frequently refer to Space Telescope oriented applications, this is meant as an example only; the development has just begun, and the design has not been finalized (Albrecht, 1981)

2. DATA

Image-type data usually come from panoramic detectors. These are devices which transfer an intensity distribution into an array of digitally represented data points. Already, such detectors are in widespread use and they will become more common with evolving technology and lower prices.

But even data obtained with the more conventional technique of the photographic process quite often are available in digital form. Microdensitometers are being used to digitize astronomical plates. Aside from the fact that data generated by two-dimensional electronic detectors are linear with intensity and data obtained by digitizing a

C. Jaschek and W. Heintz (eds.), Automated Data Retrieval in Astronomy, 87–97.

photographic plate are recorded in density units, there is no difference in principle between processing data from two-dimensional detectors and processing microdensitometer data.

Data in such sets consist of different elements. The data points of the image itself are usually referred to as pixels or picture elements. In addition to the pixel data, there is additional information needed to interpret the pixels. This part of the data set is known as the header. Note that there is no difference between pixel data and header data as far as their presence in the data set is concerned.

Other data logically belonging to the header data are the engineering data. Engineering data are not of immediate astronomic interest but rather refer to events on, for example, a spacecraft. The engineering data might become astronomically relevant once they indicate situations which affect the image data. A recent example is the temperature of the onboard computer of the IUE satellite.

There are auxiliary data, for instance, comments given by the observer at the time of measurement. Auxiliary data can also be accumulated during the processing of the data. In some systems such auxiliary data are collected in a trailer file. Ideally, the trailer file is open-ended, so all additional information can be accommodated.

The pixels within an image are usually structured in some approximation of a rectangular coordinate system. In theory, it could be a system other than a rectangular one, but for technical reasons, this is the system most often used. Within such a rectangular coordinate system the pixels are arranged in lines, lines are grouped to frames, and sometimes frames are grouped to cubes. There is no theoretical limit to the dimension of a data set, but for practical reasons, data dimensionality ends usually at three.

Examples for data cubes are: several two-dimensional frames, stacked in time. Another case, occuring very often in radio-astronomy, is a number of two-dimensional frames, stacked in frequency, to examine velocity distributions in a field.

Data coming from the detector or a microdensitometer are usually represented in integer notation. For further data analysis it is useful to convert the data into floating point notation. Although this is somewhat more costly in terms of storage and processing speed, it more than pays for itself in terms of ease of programming and safety during the reduction. Anybody who has ever divided a heavily exposed flat field frame into a low signal data frame will appreciate that.

A serious problem in image processing is the quantity of the data. A typical frame of 500 x 500 pixels holds a quarter million data points. As we begin to arrange many such frames into cubes, the data quantity grows by orders of magnitude. It is obvious that efficient

strategies have to be used for retrieving such data from a storage medium.

Most data are stored on magnetic tape as the result of a data acquisition process. For data analysis, they have to be transferred to a faster medium, which today is the magnetic disk. On the disk, the data have to be arranged in a structure convenient for retrieving them and in a format suitable for analysis. Note that there are two different data formats: an internal format and an external format. The internal format is the representation of the data on the system bulk storage. Different strategies can be used here depending on system design. Header files can be connected with the image file or kept separate. Auxiliary files can be kept in image format or in the operating system provided format. The disk can be structured in block format or data can be arranged in some kind of hierarchy.

Completely independent of the internal format is the data representation on an external storage medium, for instance, a magnetic tape for transporting images from one system to another. An agreed upon and already widely used standard is the FITS (Flexible Image Transport System) format (Wells and Greisen, 1979). For anybody transporting images, I strongly recommend adhering to the FITS standard. Not because it is an extremely elegant or philosophically pleasing design, but because it works and it is already supported by several major institutions.

3. DATA ANALYSIS REQUIREMENTS

A data analysis system, such as the one designed to reduce the data produced by the Science Instruments of the Space Telescope, must be designed in such a way as to, in principle, allow the full exploitation of the data. This is not to say that at the very beginning the system has to be capable of analyzing all data in all possible ways, but it means that the system design has to allow for later upgrading of the existing software. Such improvements are necessitated both by the fact that we learn more about how to treat the data as we gain experience with the new instruments, as well as by developments in the field. It is therefore imperative that the design of the system does not preclude later updates of the baseline.

There also has to be a healthy margin of capacity. If the system resources are only sized to handle the requirements of the baseline system, very soon user demands will outgrow the hardware capabilities of the system. This usually results in efforts to circumvent hardware limitations by software. Not only does this make the system much more complicated and much harder to maintain, in the long run, it will also be more expensive in terms of effort and manpower. Today, hardware has become very affordable and no attempt should be made to economize on hardware.

One of the major requirements of any data analysis system is that it must be user-oriented and problem-oriented. Especially at visitor-oriented institutions, the scientist cannot be expected to familiarize himself with rules dictated by and tailored to a particular computer system within the short time he spends at the observatory. In other words, the command language must be as close as possible to plain English, and the individual commands must be meaningful in the context of astronomy.

The data analysis system also has to be interactive. This means that it must be possible for the astronomer to explicitly initiate every step in his reduction procedure and check the result in near real time. Depending on these intermediate results, the astronomer will decide on and initiate execution of the next step. This procedure makes it possible for the astronomer to determine a reduction procedure approprate for his data.

Of course, interactivity also has disadvantages, especially when dealing with image data: given the amount of data points, "near real time" can be up to several minutes, depending on the operations performed. Interactivity becomes dangerous as soon as machine reaction time exceeds the human attention span, because the user will engage in other parellel activities, ultimately losing control over the reduction procedure.

Another disadvantage of interactivity is that, by definition, it requires the physical presence of the user during the reduction procedure. While this is desirable during establishing a proper procedure, it will slow down routine data reduction to an unacceptably low throughput rate. To remedy that, it must be possible, after interactively establishing the best reduction procedure, to switch to a mode of processing, in which the system executes (loops through) a set of reduction procedures for a large number of data frames. This capability is called batch-processing, or macro-processing.

A key issue in every data analysis system is the system documentation.

Not too long ago, the typical scientific program did not even have comments within the program, and there certainly was not any kind of high-level documentation. This situation has somewhat improved, but it is still far from ideal.

In addition to the documented source code, there have to be at least three different manuals: The User's Manual, which is designed to tell the user (the astronomer) how to operate the system, and, in particular, exactly what kind of operation is performed on his data; there has to be an Application Programmer's Manual, explaining how to write, maintain, and modify application programs for the system, how the data are stored on disk, and how to invoke system services; and finally,

a System Programmer's Manual is required, explaining matters like the system architecture.

It is imperative that the documentation be maintained along with the corresponding programs, so no gap will develop between the software and its documentation. For the documentation to keep pace with rapidly developing software, this means that word-processing and self-documentation techniques have to be employed.

Another form of documentation which has to be available in an interactive data analysis system is on-line documentation. This form of documentation is usually referred to as the "Help" function. Such a system feature allows the user to ask for help and explanations from the system at any time during the analysis session. A well-designed help function, guiding the user through the system, eliminates the need to consult the printed version of the User's Manual to a high degree and can make the data analysis much more efficient.

Information also has to be provided in a different area: on the scientific background of the system operations. Not only is it important to know how to operate a program to determine the instrumental magnitudes of stars in a two-dimensional frame, for the scientist it is even more important to know which method (algorithm) is used to determine these magnitudes. In other words, application programs of a data analysis system should not be used as black boxes. Of course, this is only possible to a certain degree--it is, after all, not possible to understand just what every single statement in every single program does. However, sufficient information about the program has to be provided to change the "black box" at least to a "gray box." Only in this way will it be possible for the scientist to assume responsibility for the scientific results derived from his data.

4. SYSTEM SOFTWARE

In the case of a computerized data processing system, there are usually two levels of system software to consider. The first level is the host operating system. This operating system is usually vendor-supplied and consists of an ensemble of programs which control and supervise the computer. This host operating system maintains a directory of files on the disk, interprets and executes user commands and provides a number of system services. It is called the host operating system because it provides the environment in which the actual data processing system works. While the host operating system is computer oriented and the commands it accepts are tailored very much to the computer system, the actual data processing system software is much more oriented towards the actual application. The structures of both the host operating system and the data analysis system are very similar. Both consist of a central program, which understands and interprets commands submitted for execution by the user. These commands are executed by invoking other programs known as application programs (Fig 1). The application programs are loaded into the central memory from

the disk and exit after execution. Control is then transferred back to the command decoder (Fig 1) There are two reasons for having a data analysis subsystem residing within a host operating system environment: one reason is that the commands of the data analysis subsystem can be made much more understandable to the user, the astronomer; the other reason is that in this way the data analysis software subsystem can be made transportable between computers using different host operating systems.

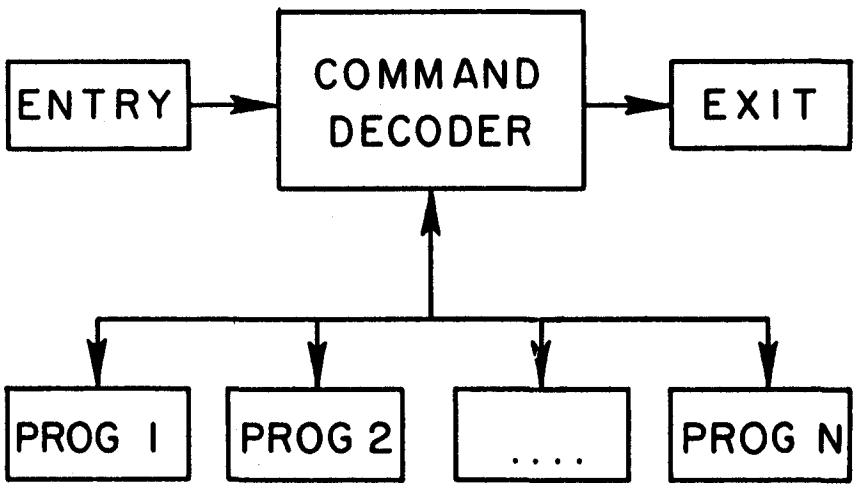

FIG. 1: Overall System Structure

A similar consideration applies to the application programs. As indicated in Figure 2, the central part of an application program consists of the algorithm, for example, a clever way of obtaining a stellar magnitude in crowded fields. This algorithm needs connections to the outside world. User parameters, for instance stellar coordinates, have to be transferred to the algorithm. Access to the data base where the pixels are stored has to be provided for the algorithm. The algorithm has to have a library of mathematical and astronomical routines available for the program. Finally, an interface must exist between the algorithm and the host operating system. This interface provides the algorithm with access to system services of various kinds, for instance a system time service, but also to more complicated system services like disk I/O. This modular structure of the application programs ensures transportability of such programs between different data analysis systems. It is also helpful during the development of the programs. Modular application programs with minimum

https://doi.org/10.1017/S0252921100082737 Published online by Cambridge University Press

interaction to other application programs are easy to develop and easy to debug. Communication between the different modules is done by clean and well defined interface routines (Allen and Ekers, 1980).

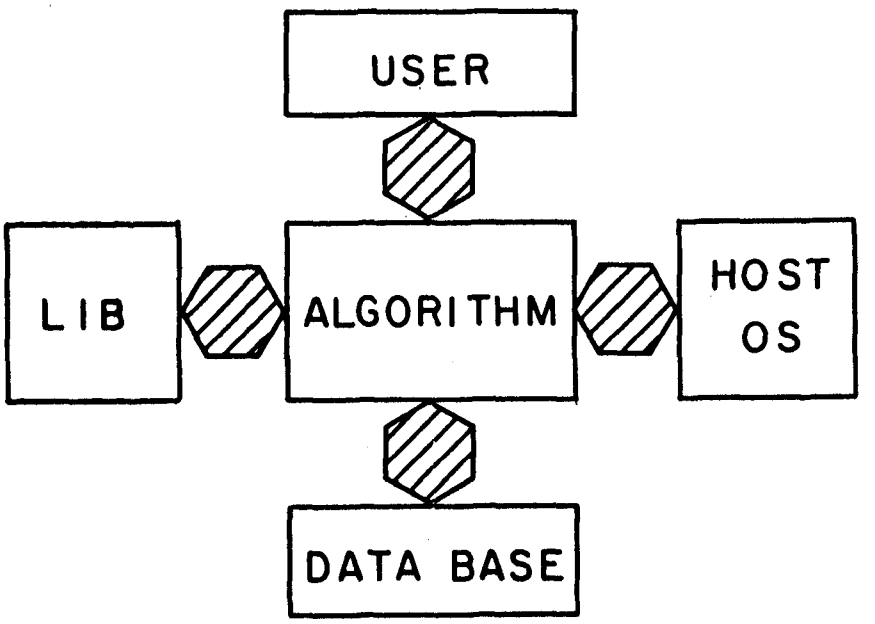

FIG. 2: *Application program structure. The shaded boxes indicate software interfaces.*

There also has to be communication between different application programs. Following the previous example, after determining a stellar magnitude in one application program, the brightness has then to be communicated to a different application program for use in further processing. Such inter-program communication can be provided in the form of communication files and system variables or stacks.

While the internal structure of the data analysis system and the application programs define the way the system works, this is usually not visible to the user. The user working at the terminal is confronted with a completely different aspect of the system, namely the command language. The syntax of the command language determines the way the user has to communicate with the data analysis software system. The command language syntax also determines to a large extent how easy or how difficult it will be for the user to actually work with the data analysis system. As can be seen in Figure 1, the command language syntax is defined by the command decoder. The communication between the command decoder and the application programs is completely independent of the command language syntax. It is therefore possible to communicate with a set of application programs through more than one command decoder, defining more than one command language. Another possibility

https://doi.org/10.1017/S0252921100082737 Published online by Cambridge University Press

is to have command decoders who understand more than one command language.

On the other hand, it is possible to use application programs from different systems as long as their respective software interfaces are supported. It is therefore conceivable to have a hypothetical system consisting of application programs from many systems and invoke them in the command language of a completely different system. A data analysis system following these considerations is currently under development for the analysis of data generated by the Faint Object Camera of the Space Telescope.

5. DATA BASE INTERFACE

As already discussed in the first chapter, data ready for processing usually reside on a magnetic disk. Access to the magnetic disk is provided through the low level device driver interface provided by the host operating system. Such low level interfaces are very specific for the individual computer and the individual hardware configuration. Using such low level interfaces in an application program therefore makes them non-transportable and also difficult to read and understand: low level interfaces access the disk in the hardware specific form of sectors, blocks, or cylinders. For the application program, a high level interface is desirable, accessing the disk in the astronomy oriented terms of pixel coordinates.

High level data base interface routines can also be standardized and seperated from the computer dependent low level interfaces by layers of communication routines, thereby making the applications program and to some extent, even the high level interface transportable between different computers. Standardization and transportability are important considerations in an environment where software is being developed by more than one group. For the Space Telescope Science Data Analysis System, software will be provided by the teams developing the different Science Instruments for the telescope. At the Institute, we are currently developing a system of standardized interfaces so the software developed in different places is usable and useful to groups working at other installations.

Standardization of the data base access is of particular importance for the software development for the Space Telescope: the data generated by the majority of the Science Instruments are image data, as discussed in Chapter 2. All programs have to access those data. The access has to be very efficient, so as to optimize throughput. To fully utilize software developed at other institutions, in particular software developed at the Teams developing the Science Instruments, it is clearly necessary to achieve standardization as early as possible.

Being able to use software that was developed elsewhere also implies that the software be transportable between different computers. It is clearly not possible to develop software that will run

on all computers without modifications. However, following a few rather basic rules, it is easily possible to develop software that will run on many computers with only small modifications. Following those basic rules also has another beneficial byproduct: it forces a modular, structured design, which makes the software easier to develop, to understand, and to maintain.

Adequate software maintainance is a key issue, especially for developing software in a research environment, where input comes from many different sources, and the problems to be solved themselves are changing. A program, which is, for instance, developed at the Wide Field Camera Instrument Team, even though it may be transportable, is not necessarily usable for Faint Object Camera Data: the WF/PC generates 12-bit data, while the FOC generates 16-bit data. Even after the program has been generalized to handle data from both instruments, it will have to be changed again to follow suggestions of the astronomers, who have been using it.

It is also clear that such changes, even though they might be beneficial and actually improve the program, have to be implemented with considerable care. Evidently, it is impossible to replace a particular program by another program, even a better one, just as it is impossible to replace the photomultiplier in a photometer in the middle of an observing run.

Such changes in the software have to be made in a carefully controlled fashion. First, it has to be made certain that the requested change to the program does not jeopardize its performance otherwise (e.g., increasing processing time by an order of magnitude). After making the change and implementing it, the new version will have to be tested, both to make sure that the change produces the desired results, but also to verify that other results produced by the program have not been impacted by the change. Finally, the program documentation has to be updated and the program is released to the users. This process is called Configuration Control. It is one of the most important aspects of a large software development project.

Applying these considerations to the problem of data base interfaces, we find the following situation: at the device driver level, the interface is machine-oriented and thus difficult to standardize and certainly not transportable. At the highest level, which is visible to the applications program, standardization is an absolute necessity. This leads to the concept of interface routines consisting of several layers of software between the standardized front end and the machine dependent device driver. Using this concept, it is even possible to maintain transportability of the uppermost layers. Figure 3 shows this concept as used in the Tololo-Vienna Interactive Image Processing System. On the highest level, the user accesses the image like a large FORTRAN array, using an image name and the pixel coordinates as subscripts. This makes it very easy for the astronomer to write programs performing operations on pixel data.

```
PDP-11

FRONT END:      IPIX=IMAGE (IX,IY)
STRING I/O:     CALL @READ (LINE,IBUF,IWORDS,IERR)
BLOCK I/O:      CALL QIO (.....)
DEVICE I/O:     Not accessible

HARRIS-DATACRAFT

FRONT END:      IPIX=IMAGE (IX,IY)
STRING I/O:     CALL @READ (LINE,IBUF,IWORDS,IERR)
BLOCK I/O:      CALL IFXXYY (.....)
DEVICE I/O:     Not accessible

VAX, PHYSICAL I/O

FRONT END:      IPIX=IMAGE (IX,IY)
STRING I/O:     CALL @READ (LINE,IBUF,IWORDS,IERR)
BLOCK I/O:      CALL SYS$QIO (.....)
DEVICE I/O:     Not accessible

VAX, VIRTUAL I/O

FRONT END:      IPIX=IMAGE (IX,IY)
STRING I/O:     } Mapping
BLOCK I/O:      } Hardware
Device I/O:     Not accessible
```

FIG. 3: Layered interfaces for different host computers.

As can be seen, in this particular interface even the second level is standardized, making the complete first layer transportable. Contact with the host operating system is established only at the third level, where system services are being used to access the disk. This is not applicable to virtual I/O on the VAX, where the VAX-specific mapping features are being used in the first level.

The question of the efficiency of such interfaces has been raised. Clearly, the fastest way of accessing the data is to skillfully use the machine-dependent features of the device driver. However, it is also clear that most astronomers are not willing to write programs at this level.

On the other hand, there is undoubtedly some overhead associated with the high level access of the data base, sometimes slowing down processing considerably. The solution we have adopted is to make the lower layers also accessible to the application program. Thus, if it becomes necessary to speed up a program developed by an astronomer, using front end calls, a system programmer can replace them by faster, low level access calls.

https://doi.org/10.1017/S0252921100082737 Published online by Cambridge University Press

6. CONCLUSION

The recent advances in hardware technology and their application to astronomical data acquintion as examplified by panoramic detectors and orbiting observatories, have brought about the necessity to refine the software tools needed to analyze the data. These tools exist, however, astronomers have not quite learned how to use them efficiently. It must be emphasized that the long term return on investments in hardware and manpower ultimately depends on the quality of the software used for the analysis of the data.

References

Albrecht, R. 1981, Proceedings of the Space Telescope Science Data Analysis Reconnaissance Meeting. ST ScI, Baltimore, MD.

Wells, D., Greisen E., 1979, FITS: A Flexible Image Transport System. In: Image Processing in Astronomy, G. Sedmak et. al. (Eds). Osservatorio di Trieste, Italy.

Allen, R.J. and Ekers, R.D. 1980, Creating Durable Software in a Research Group Environment. In: IUE - Data Reduction, Weiss et. al (Eds.) Vienna, Austria.

https://doi.org/10.1017/S0252921100082737 Published online by Cambridge University Press