# The Hardware and Software Support for the MRSP

**Dieter Teuber**
Astronomisches Institut
Westfälische Wilhelms-Universität
Münster, F.R. Germany

**Abstract**

The Muenster Redshift Project (MRSP) described by Horstmann (1988) and Schuecker (1988) relies on an arrangement of hardware and software which is referred to as the Astronomical Data Analysis System (ADAS). In this paper the hardware will be briefly introduced and the support software GAME will be discussed.

## 1 The hardware

Direct Schmidt plates from the ESO/SRC-Survey and corresponding objective prism plates are digitized with the improved microdensitometer PDS 2020 GM plus, using a sampling width of 15$\mu$m. Each plate yields approximately one gigabyte of data. These data are processed with the 32 Bit minicomputer PE 3220 under the operating system OS32. The respective application programmes either reduce the data online or store them on tape or disk. *Fig. 1* shows the hardware configuration currently in operation.

## 2 Designing software for the ADAS

The software of most ADASes has *grown* through the years driven by the (momentary) needs of interactive users causing incompatibility among application software. In order to avoid inconsistencies the MRSP software has been accompanied by a **designed** support software system from the beginning. For a successful design a thorough analysis of the data reduction process has to be carried out.

Processing of digitized data from wide angle plates results in hundreds of thousands to millions of object images. These images have to be detected, calibrated and classified by an expert system. The exchange of expert knowledge is the major purpose of a conference like this one. But how can we take expert knowledge to the computer and have it applied to the data?

A normal way to cope with complex problems is to break the task into steps which are less abstract. *Fig. 2* illustrates from left to right how a task is transferred to less abstract algorithmic levels until the computer hardware is reached. The bottom to top course of the enquiry leads from the sampled image to more abstract object levels.

In the astronomer's mind an (astronomical) *world model* is formed. From his current world model he derives questions which he tries to answer through astronomical ob-

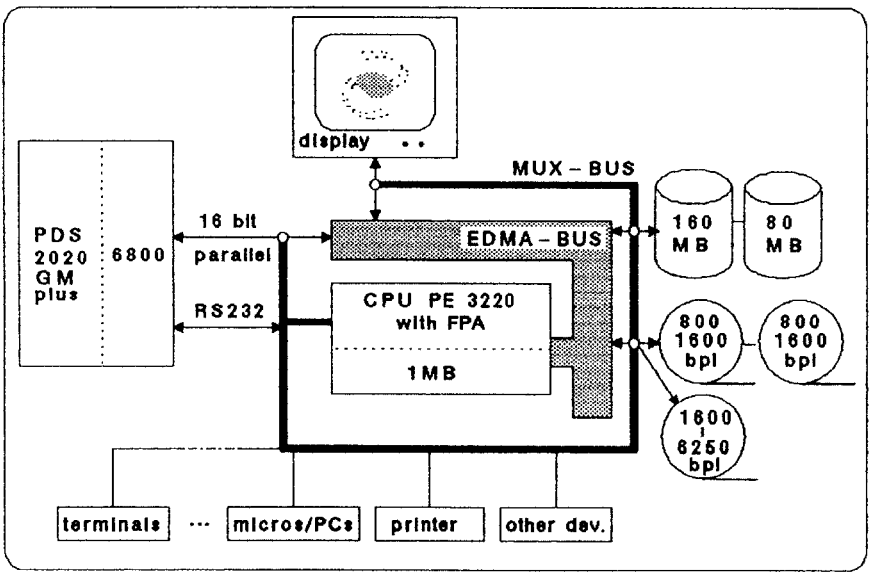


**Fig. 1.** The hardware configuration of the AIM.

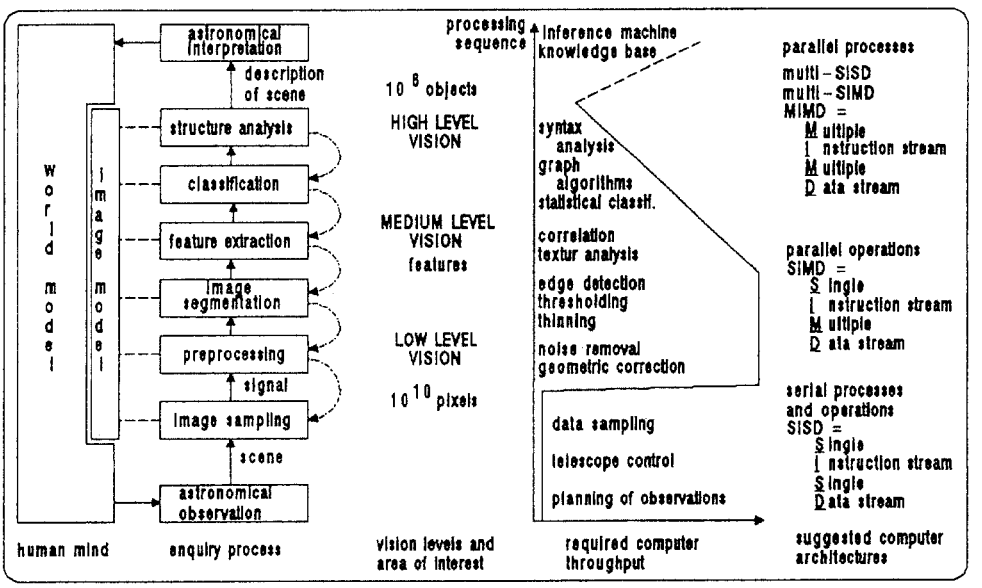


**Fig. 2.** How to take MRSP expert knowledge to the computer hardware (for explanation see text).

servations. He then processes and interprets the observed data following procedures which depend on his observational techniques. The answer to 'What is an image or object?' is given by the astronomer himself – often intuitively. As soon as he hands the decision to some device, he has to supply also a model which decribes the image of an object. This *image model* influences the whole analysis process. If an expert system requests a new observation, it will also influence the observation and the sampling process.

An enquiry may be carried out in various ways. In *Fig. 2* a potential sequence of analytical steps is shown. It should be noticed that in the course of enquiry these steps are often repeated in iterative loops. The steps can be grouped into *low*, *medium* and *high level vision* which are characterized by their area of interest: pixels at low level, image structures at medium level and astronomical objects at high level.

For each step algorithms must be found and coded in a programming language. From the programmes the resulting demands on support software and hardware can be determined. The two right hand columns of *Fig. 2* concern the coding and execution of data reduction procedures. Typical procedures and the required computer throughput are shown as they are used during each step of enquiry. For observations and subsequent steps not much computing power is needed until the observed scene is transferred to the computer. In the pixel regime array processors are the most suitable devices. Coming closer to the object regime, procedures can be divided into parallel processes, *i.e.* several procedures applied to one object or the same procedure applied to several objects. These processes run on multi–computers or transputer arrays.

Having looked at the process of enquiry, the model of a suitable ADAS can be outlined according to the demands of the user. Within the ADAS the user needs a facility to handle his procedures, data and devices in a way which reflects the current abstraction level of his work. But a high abstraction level requires a high degree of automisation. This statement shall be made evident by some examples.

A request from the user may expand into complex operations and automated sequencing of programmes becomes necessary. The facility shall determine the next action evaluating the status of the data and of the ADAS. But the facility should not only give support in local operations. New data often become more valuable when combined with existing data. This requires access to other computer systems, so that remote login and data retrieval from foreign data bases must be provided. Thus a request may lead to activities ranging from reading a parameter from a local storage location up to remotely running a programme on a foreign computer and transferring the result back into the application programme via communication networks. The details of the various routes of access to the data are of no concern (except for cost) to the user and should be hidden.

The amount of scientific data to be handled will also influence the structure of the facility. Large amounts of data which will be accessed by various selection criteria demand for organising the data by a data base system. For each data aggregate a log of its processing history must be recorded.

To avoid unwanted activities of the automated system the user must be informed, not only of the results, but of all actions taken during processing his request. The user must be able to interrupt and abort processing. This, of course, demands for a mechanism to return to the previous state. Problems like these are known from transaction systems.

The requirements of automised data analysis and those of interactive data analysis have both to be met for the MRSP. The approach introduced in this paper is to administrate the flow of information by maintaining and evaluating descriptions of all subjects and objects participating in the data analysis process:

Data analysis on high abstraction levels is achieved by embedding the participants in an administrative software environment.

## 3 GAME: The internal structure

A Generic Applications and Monitors Environment (GAME) has been developed and implemented at Muenster. According to the recommendation of the WGCAS (1983) the implementation language is FORTRAN 77. GAME forms the basis of the work in interactive and batch mode (= natural intelligence, NI modes) and expert system mode (= artifical intelligence, AI mode). GAME is the link between all parts of the ADAS, hardware and software. GAME screens the users and applications from the specific properties of the actual computer system and supports developments and data handling on a more abstract level. Some principles of the GAME concept were outlined by Teuber (1985). GAME matches the features attributed to virtual operating systems as defined by Tody (1987). *Fig. 3* illustrates the interfaces of GAME to the participants and the modular structure of the interior of GAME.

The functional units of GAME are called administrators (ADM). ADMs are sets of routines which offer support in organising data and programmes, but do themselves no data reduction. Each ADM possesses its proper data structure which is not visible outside the ADM. These data structures are optimized for the specific needs of the ADM operations. ADMs can be compared to *modules* as they are defined in MODULA-2.

GAME offers services to four sides. To the interactive (astronomical) user a session monitor offers individual communication methods. Application interfaces provide data access, environmental control and check points for application programmes. A report facility supports the system manager in maintaining a functioning ADAS. The operating system is supplied with the appropriate system calls.

To achieve a high degree of portability all dependencies on operating system and hardware are resolved at the bottom layer of GAME. The interfaces to the operating system contain all system calls to perform intertask control and communication, to steer the file system, to translate symbols of the operating system into GAME symbols and vice versa, and to perform the data transfer. Other system dependencies like language extensions are isolated in a separate library.

All references to the outside world issued by an application programme are made

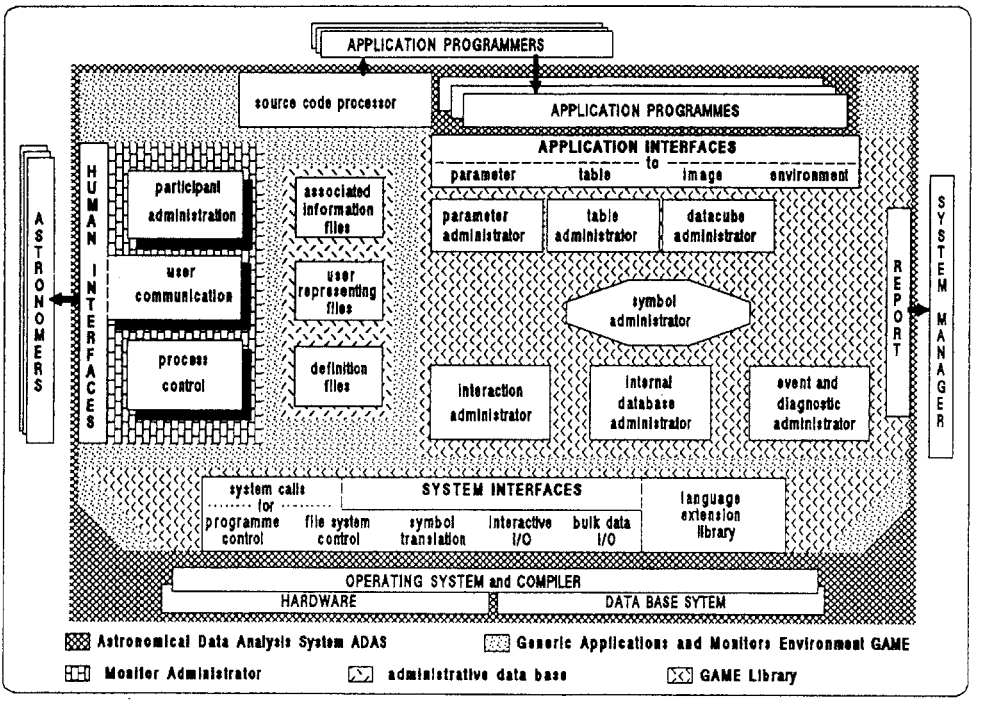


**Fig. 3.** GAME (interiors and interfaces).

using keywords. In GAME keywords may be connected to data aggregates or may translate into operations. The latter case leads into the domain of object oriented processing. Keywords may be regarded as messages sent to (data) objects which respond by executing methods.

## 4 GAME: Organisation of data

*Data storage* is organised in a kind of hierarchical data base which consists of so–called **structure trees** (not to be confused with simple tree structures). A structure tree is made recursively of nodes which themselves are structure trees, *i.e.*, each node contains in itself a structure tree, which is independent of the predecessor. A branch of the tree terminates with a leaf containing user (scientific) data together with a description.

*Figure 4a* shows an example of a structure tree. The root element is used to link the tree into the current GAME environment. In the example shown here the root opens a directory of contiguous type. When the user supplied path specification selects the middle element, the path continues into the next directory. When the element to the right is selected, the path leads into a list of data elements from which one element is chosen. The user has to specify only the *path* to find this value, he does not need to know the storage structures which have to be passed. The data could be stored in some different data structure, but addressed with the same path specification.

The path may be either a *key* to a certain location, as in the example, or a *directive* to search for a number of instances. It is apparent from the sketch of the administrator in *Fig. 4b* that the menu-like character of the administrator programme makes it easy to add new structures to the system. A dynamic path linker enables the redirection of a user specified path into another (part of a) structure tree.

Using the appropriate interface calls, the application programmes may store and retrieve data formatted as *image*, *table* or *parameter*. Requests for parameters may be redirected to the list and command channel of a process. This is the only way for an application programme to communicate directly with the user (expert system or human). *Image display* and *graphics interfaces* are available, but at present not yet fully integrated into GAME. A revised version of the respective software will meet existing standards such as IDI (Terret 1986) or GKS or AGL (Fini 1986).

*Parameters* are the simplest data units. A parameter consists of a name which is the key (when referenced within a structure) to a storage area which contains a control and a data segment. The control segment holds the description of the structure in which the values are arranged and the format of the coding. The structural description permits to handle the parameter as a single value, array, stack, ring buffer and/or menu.

*Tables* and *catalogues* of various formats may be processed using a corresponding definition file. Once the file is created all GAME users are able to access the respective catalogue.

An *image* is a compound format which uses a header made of parameters and tables to describe the data in the *datacube*. Several images may be packed into one file and

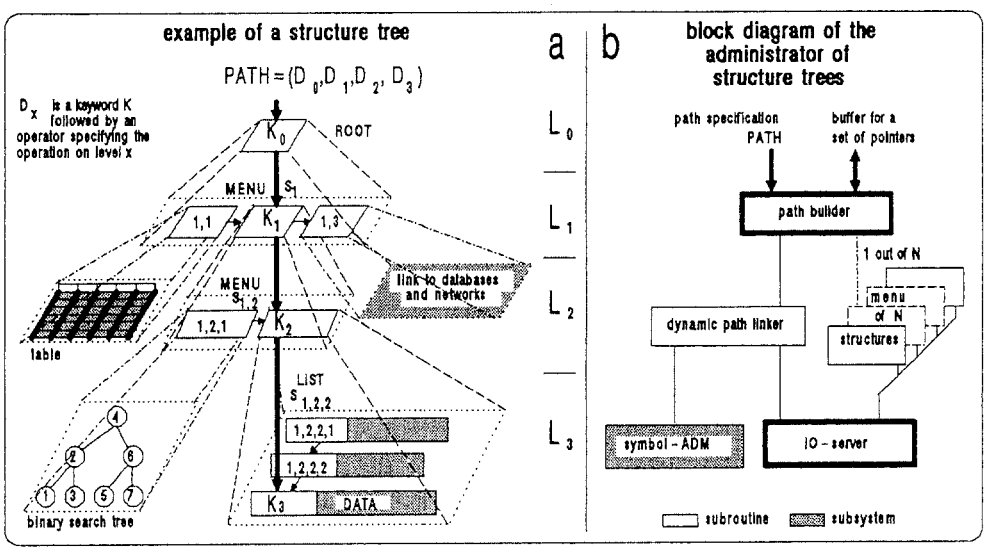


**Fig. 4.** a) a structure tree
b) scheme of the administrator of structure trees.

interrelated by a structure tree. In its simplest form a structure tree may serve as a list–structured or a hierarchical directory. It may be extended to form a kind of hierarchically organized database. Images written in FITS format according to Wells *et al.* (1979) may be processed on–line, *i.e.*, they do not need to be preprocessed by a FITS–reader.

As has been shown by Teuber (1988a) the application data aggregates are built from internal data structures of GAME. Those internal structures are transposed to the operating system and the hardware structures by a small number of system interfaces.

## 5 GAME: Organization of activities

When more than one process is active in an environment, the access to resources such as microdensitometers, colour displays or transmission lines has to be organised to avoid scrambling of data or deadlocks. In terms of computer science: a monitor is needed. For GAME the **Symbol Administrator** serves as the synchronizing mechanism. The data structure inherent to the Symbol ADM is the Symbol Queue. Every participant (=subject and object) in a data analysis process is assigned a slot in the Symbol Queue. Each slot has a descriptor for easy access by the human user, but is uniquely identified by a numerical index on the Symbol Queue to GAME. Multiple links may be established between the slots of the Symbol Queue by a set of pointers. One of them is used to form processes as a linked list.

The GAME organisation becomes visible to the application programme only through the handle returned after the respective participant has entered the reduction process. The environmental interfaces enter and remove a programme or a data aggregate onto/from the Symbol Queue.

The user may modify the response of GAME to requests over a wide range using the **event administrator**. The event mechanism is activated as soon as the event slot is included into the Symbol Queue. While obeying a command, GAME often encounters states, where continuation is possible in more than one way. By setting up the event list properly, the user can induce GAME to act in a specific way, instead of defaulting. The event administrator can also be used to check the processing history of a data aggregate or the user environment. More details are given by Teuber (1988b). The ADM of events also supports the detection of errors inherent to GAME.

The most complex unit is the **administrator of session monitors**. Based on definition files stored in the administrative data base, the monitor ADM generates the actual session monitors. A session monitor interfaces to the user via a communication method (language, menu, icons), supervises the processes initiated by the user and communicates results and status information to the user. The functional details of each monitor can be varied over a wide range by setting up the corresponding definition file. Thus the monitor ADM possesses properties of a user interface management system (UIMS, Green 1985). The monitor ADM is supported by the **interaction administrator**. Depending on the environment of the process, the interaction ADM either generates short prompts to the user or links the application programme to the monitor.

The aspects of registering application programmes with GAME are briefly described by Teuber (1988a).

## 6 Concluding remarks

When GAME supports an expert system, instructions to the system and messages from the system can also be considered as data (procedural data). An expert system may generate instruction to the ADAS and interpret messages from the ADAS. The session monitor can function either as an interactive user interface or as user port and ADAS interface to the expert system. Running an application programme in NI mode without session monitor is a matter of *convenience*. The session monitor and the related environment becomes a *necessity* when one proceeds towards an expert system. The definition (file) of such a monitor may well be elaborated using an AI language such as LISP.

GAME was originally developed on a Perkin Elmer 3220 computer under OS32. The above text refers to GAME version 3.0 which is scheduled to be ported to a UNIX-based computer system in 1988.

## Acknowledgements

I thank Markus Tacke and Stefan Grefen who joined the Institute to support the MRSP from the side of computer science. They have begun to investigate the problems related to software portability when working with parallel processes and parallel processors and when using UNIX in a real time environment. Acknowledgements are also given to Randolf Budell who was instrumental in implementing the ADATA storage format and gave support in hardware maintenance.

## References

Fini, L., 1986. *ASTRONET Graphic Library Reference Manual*, Version 2.1.

Green, M., 1985. In *Workshop on User Interface Management Systems, Proceedings*, ed. Pfaff, G.E., Springer, Berlin, p. 9.

Horstmann, H., 1988. *These proceedings*, p. 111.

Schuecker, P., 1988. *These proceedings*, p. 142.

Terret, D., Shames, M., 1986. *An Image Display Interface for Astronomy (draft)*, Version 0.4, STScI, Baltimore.

Teuber, D., 1985. In *Proceedings of the International Workshop on Data Analysis in Astronomy*, eds. DiGesu, V., Scarsi, L., Crane, P., Friedman, J.H., Levialdi, S., Plenum Press, New York, p. 235.

Teuber, D., 1988a. In *Astronomy from Large Databases*, eds. Murtagh, F., Heck, A., ESO Conference Proceedings, p. 261.

Teuber, D., 1988b. In preparation.

Tody, D., 1987. *The IRAF Data reduction and Analysis System.* In *IRAF System Handbook*, **3A**, Natl. Opt. Astr. Obs.

Wells, D.C., Greisen, E.W., Harten, R.H., 1981. *Astr. Astrophys. Suppl.*, **44**, 195.

WGCAS, 1983. *Working Group for Coordination of Astronomical Software*, ESO, Garching, unpublished.