SCIENCE AND ENGINEERING RESEARCH COUNCIL
RUTHERFORD APPLETON LABORATORY
Starlink Project
Chilton
Didcot
Oxfordshire
England
OX11 0QX

Proceedings of the 1989 ADAM Workshop

Editor: A J Chipperfield

16th January 1990

Contents

Part I

Overview

Chapter 1

Introduction

1.1 The ADAM Workshops

ADAM is now a major software project; it provides a fully integrated environment for both data reduction and data acquisition. It is being used in Hawaii, Australia and the Canary Islands, as well as the UK, and has been adopted by Starlink as the environment in which Starlink data reduction software should run. One of the most remarkable things about ADAM is that it has been developed as a co-operative effort between groups that are spread across the world. Although the initial system came out of RGO, and ROE provided by far the major effort in designing and implementing the VAX version, various parts of what is now regarded as 'ADAM' have also come from other establishments. Co-ordinating a project being developed in this way is not an easy job, but the somewhat varied parentage of ADAM – although sometimes an administrative nightmare – is also one of its strengths; it is not a system developed in one place to serve the specific needs of that one place.

One way in which this development is co-ordinated is by a series of workshops. These have taken place at about 18 month intervals since the first one in late 1985. The workshops are attended by people actively developing and/or making extensive use of ADAM, and provide a forum for detailed discussion of the problems in the current system and plans for its extension.

The 1989 ADAM Workshop was held at Cosener's House, Abingdon from 3rd to 7th July 1989. An 'Open Meeting' was held on Friday 30th June at RAL to enable members of the Starlink community to provide input to the Workshop discussions.

Before the previous workshop, in Hawaii, a trend had started to emerge for different establishments to plug the gaps in ADAM (which at the time was missing a number of important facilities) with local solutions. The Hawaii Workshop consolidated these local extensions, adopting some and rejecting others. As a result, ADAM, as reviewed by this third workshop, was a much more complete and uniform system, and it was possible to start to look in detail at the various enhancements that were still needed; in particular, to make it efficient as a data reduction environment.

This document summarises the conclusions and actions arising from the Workshop and presents brief reports on the discussions, prepared, in most cases, by the session chairmen. Part III consists of documents submitted, prior to the Workshop, for consideration during it; their content was not necessarily endorsed by the Workshop. Part IV consists of those reports and proposals, arising out of Workshop action items, which have already been submitted.

1.2 Participants

William Lupton	AAO	CBS%AAOEPP::WFL
Keith Shortridge	AAO	CBS%AAOEPP::KS
Tony Farrell	AAO	CBS%AAOEPP::TJF
Jeremy Bailey	JAC	CBS%JACH::JAB
Bernard McNally (BVM)	JAC	now REVAD::BMC
Dennis Kelly	ROE	REVAD::BDK
Lewis Jones	RGO	GXVB::LRJ
Jonathan Burch	RGO	GXVB::JMB
Nigel Houghton (NRH)	La Palma	RGVAD::LPMAIL (Subject Nigel Houghton)
Bob Vallance	UofB	BHVAD::RJV
Peter Allan	UofM	MAVAD::PMA
Malcolm Currie (MJC)	Starlink	RLVAD::CUR
Alan Chipperfield	Starlink	RLVAD::AJC
Dave Terrett	Starlink	RLVAD::DLT
Rodney Warren-Smith	Starlink	RLVAD::RFWS
Patrick Wallace	Starlink	RLVAD::PTW
Mike Lawden	Starlink	RLVAD::MDL

Also attending some sessions were:

Jo Murray	Starlink	RLVAD::JM
John Sherman (JCS)	Starlink	No e-mail
Dave Giaretta	RAL	STADAT::DLG
Monica Kendall	RAL	STADAT::MLK
Clive Page	UofL	LTVAD::CGP

Mail addresses from Starlink machines are given in the third column.

Participants may be referred to by their initials, which in most cases are the same as the username in the mail address. Where this is not the case, initials are given in parentheses after the participant's name.

1.3 Abbreviations and Glossary

AAO	Anglo-Australian Observatory
AAT	Anglo-Australian Telescope
ACT	An application-dependent routine to perform the task actions (see AED/1)
ADAMSC	ADAM Steering Committee
ADC	A library for for handling relational data (see SUN/71)
AED	ADAM Environment Description
AON	ADAM Observer Note
ASG	ADAM Support Group (Starlink funded, from April 1990)
AST	VMS Asynchronous System Trap
Asterix	An X-ray data processing applications package (see SUN/85)
CMS	DEC's Code Management System
DCL	DIGITAL Command Language used with DEC's VMS operating system
DEC	Digital Equipment Corporation

ERR	ADAM's error reporting facility (see AED/14)
Figaro	A general data reduction system (author KS) (see SUN/86)
GKS	Graphical Kernel System (see SUN/83)
GNS	Graphics Workstation Name Service (see SUN/57)
HDS	Hierarchical Data System (see SUN/92)
ICL	Interactive Command Language (author JAB)
IDI	Image Display Interface – an international standard in astronomy (see SUN/65)
IFL	Interface file (see AED/3)
INT	Isaac Newton Telescope (La Palma)
IPMAF	IRAS Post Mission Analysis Facility (RAL)
IRAS	Infra-Red Astronomy Satellite
JAC	Joint Astronomy Centre (Hilo, Hawaii)
JCMT	James Clerk Maxwell Telescope (Hawaii)
KAPPA	Kernel Applications Package (see SUN/95)
MMS	DEC's Module Management System
MON	A fast monitoring and transfer system (see AED/2 and AED/9)
MONGO	An interactive plotting program (see SUN/64)
MSG	ADAM's message facility (see AED/14)
MSP	Message System Primitive Routines (see SSN/2)
NBS	Noticeboard System (author WFL)
NCAR	Graphics package from The National Center for Atmospheric Research (see SUN/88)
Obs	Observatories (AAO, La Palma, JAC)
PGPLOT	A graphics subroutine library (author T J Pearson, Caltech) (see SUN/15)
RAL	Rutherford Appleton Laboratory
RGO	Royal Greenwich Observatory
ROE	Royal Observatory, Edinburgh
ROSAT	A joint USA, UK and West German X-ray satellite (due for launch May 1990)
RPT	A low-level error reporting facility (author Johan Hamaker, JCMT)
SCAR	Starlink Catalogue Access and Reporting (see SUN/70)
SG	Starlink Guide
SIG	Special Interest Group
SLW	Sid Wright – author of much of the original Starlink Software Environment (SSE)
SMG	DEC's screen management library
SMS	ADAM's Screen Management System user interface
SSE	The original Starlink Software Environment – superseded by ADAM
SSN	Starlink System Note
SUN	Starlink User Note
UKIRT	United Kingdom Infra-Red Telescope (Hawaii)
UofB	University of Birmingham
UofE	University of Edinburgh
UofL	University of Leicester
UofM	University of Manchester
VMS	An operating system for DEC VAX computers
WHT	William Herschel Telescope (La Palma)

Chapter 2

Summary

2.1 Conclusions and Actions

This section summarises the main conclusions and actions generated at the Workshop. More detailed discussion of the topics may be found in the reports on the sessions. It should be noted that, because most participants have numerous other duties, no timetables or guarantees were given by people accepting actions – the Workshop itself has no authority to direct effort. Nevertheless, many actions have already been completed.

Reports and proposals submitted to date in response to actions are included as appendices to this document and referenced here. Actions which were implemented in ADAM Version 1.6, released on 6th December 1989, are flagged with '[V1.6]'.

Review of previous workshop

- ADAM V2 will not contain ENGIF. (AJC) [V1.6]
- ADAM V2 will be the last major release that contains ADAMCL, MON and DIAGRAM. (AJC)

Software reliability

- Starlink should bid for quality assurance person(s). (PTW)
- The need for checking critical parts of the system is recognised and the ADAM support group is requested to coordinate this effort. (ASG)

Documentation

- The ADAM guide needs revising with, amongst other things, more sections on application packages, description of ADAM structure, ... (MDL)
- The ADAM Guide will contain short sections on all ADAM application packages. Suggested authors were: Bob Vallance (ASTERIX), Nick Eaton (DAOPHOT and PHOTOM) and Helen Walker (SCAR).
- Documents in ADAM series (AONs, AEDs *etc.*) will slowly be retired and replaced with Starlink documents. (MDL)

- Starlink documents will acknowledge the author's institution. (MDL)
- Production of ADAM data analysis and data acquisition programmers' manuals is regarded as a high priority for the ADAM support group. (ASG)
- Both ADAM and stand-alone versions of a given package should be described in a single document. (MDL)
- Programmer-level documentation is required for all packages. Such documents should have a standard format of 'philosophy followed by descriptions of routines'. (ASG)

Releases and bug reports

- ADAMSTART should display an over-all ADAM version number. (AJC) [V1.6]
- Test suites for D-tasks should be provided. (Obs)
- A public ADAM programming conference will be set up. (AJC) (Appendix I.1)
- All ADAM bugs should be reported to RLVAD::STAR. (All) (Appendix I.2)
- Proper timing tests must be performed on ADAM packages so that it will be possible to measure the effect on performance of any future changes. (ASG)
- There is a need for a definite policy on which computer systems ADAM is supported (*e.g.* on multi-processor systems?). (ASG)
- A strong effort must be made to achieve upwards compatibility of application code. (ASG)
- Proper use should be made of shareable image *major-id* and *minor-id.* (AJC) [V1.6]
- ADAM should use Starlink shareable libraries (rather than linking them into ADAM shareable images). (AJC)
- Starlink will rationalise its use of shareable libraries. (DLT)

Private versions of IFLs

- The ADAM_IFL logical name will be used as a search path (IFC then IFL in each directory). (AJC) (Appendix J) [V1.6]
- COMPIFL will support an INCLUDE facility. (AJC)
- It is no longer planned that the automatic run-time compilation of interface files should be removed.

Error reporting

- Following the inconclusive discussion, a firm proposal is required (RFWS).

HDS

- Temporary storage problem: on annul of temporary object, will delete it and truncate the file if space at the end of the file is unused. (WFL)

- Dangling locator problem: to be resolved by using a unique sequence number for each locator. (WFL)

Parameter system

- > 1 line of help text: will use the help text as VMS help library and topic specification. Detailed proposal is required. (JAB) (Appendix G)
- Abbreviated parameter keywords: will use automatic non-ambiguous abbreviation. Detailed proposal is required. (AJC)
- MIN and MAX: will support MIN and MAX responses. Detailed proposal is required. (WFL) (Appendix F)
- '\' in response to prompt: will be supported. Method agreed. (AJC)
- PAR inquiry routines: detailed proposal required. (LRJ)
- Strong typing in IFL: violations will be handled as errors. (AJC)
- User-supplied conversion routines: detailed proposal is required. (LRJ)
- HDS impact on parameter system operation: timing tests are required. (AJC)
- Conversion of responses to upper-case: use IFL keyword. Detailed proposal is required. (AJC)
- Behaviour when getting cancelled parameters: always prompt. (AJC) [V1.6]
- Behaviour of RESET: ignore 'current' in vpath and ppath. (AJC) [V1.6]
- Setting parameter state: need modified PAR_CANST proposal. (AJC)

Distributed NBS

- Produce write-up of background and proposed implementation strategy. (WFL) (Appendix K)

Tasking architectures

- Task type unification: agreed in principle. Need detailed proposal. (TJF)
- ADAM system modifications: complete and release. (WFL)
- Replacement for AZ$SNDAST: discuss with Lewis Waller. (TJF/LRJ)
- Monolith resource allocation/release: need definite proposal. (TJF)

ICL

- Double precision for scalars: agreed. (JAB) (Appendix H.1) [V1.6]
- Command line editing: save last n commands in a file. (JAB) (Appendix H.2) [V1.6]
- DCL access: allow '$' in place of 'DCL' and remove DIR command. (JAB) (Appendix H.3 and H.4) [V1.6]

- Lexical functions: provide equivalent of F$ELEMENT. (JAB) (Appendix H.5.1) [V1.6]
- File existence: provide means of testing file existence. (JAB) (Appendix H.5.2) [V1.6]
- Figaro support: use DEFUSER and modify existing FIGARO command. (JAB) (Appendix H.6) [V1.6]
- Use of wild-cards: need detailed proposal. (PMA)
- Arrays: Need definite proposal. (ASG)

Graphics from the command language

- Philosophy: preferred solution is to have low-level operations in a monolith.
- Need survey of most appropriate graphics package to support. (ASG)
- Leicester PGPLOT wrap-around package: circulate details. (CGP)

SCAR

- ADC interfaces: urgently need reviewing to assess suitability for support by the ADAM support group and to ensure that they could support different database systems. (ASG)
- Alternative interfaces: circulate details of STDB interfaces. (DLG)
- Timing: circulate comparative timing tests. (DLG)
- Application access to user interface screen: need definite proposal. (DLT)

Figaro

- Help: use DEFHELP rather than HLP$LIBRARY definitions. (KS)
- Starlink release of monolith: plan for mid-October with updated documentation but before all known problems are fixed. (KS)
- User variables: access ADAM_USER:GLOBAL.SDF directly. (KS)

ADAM V2

- Security: discuss security aspects of the ADAMNET process. (BVM/DLT)

Workstations

- Apply for grant for person to work on SMS conversion to DECwindows (with input from Starlink and all observatories). (UofE)

Instrument and telescope interfaces

- Not practicable to standardise interfaces to instruments; possibly possible to standardise interfaces to telescopes.

- Should be possible to standardise the interface between D-tasks and instrument control micros. (BDK/NRH)

Interrupting ADAM tasks

- Need definite proposal. (TJF)

Portability

- Use of %VAL can continue. A pre-processor may be needed for some operating systems.
- Should investigate availability of ADA compilers and the true implications of using ADA for all or part of an ADAM system. (NRH)
- Starlink is intending to attempt to port a limited subset of ADAM to a DECstation running Ultrix.
- A port should not be achieved by reducing functionality to the 'lowest common denominator'.

D-task fixed part

- An enhanced DTASK_ASTSIGNAL is to be provided. (BDK)
- A detailed proposal for concurrent and/or queued task actions is required. (JMB)
- A feasibility study is required on asynchronous handling of incoming commands by the D-task fixed part. (JMB)
- A facility for invoking one action from another in the same task is not necessary.
- A facility for delivering a user AST to a control task on receipt of a message from a subordinate task was decided against.

Setting default directory

- Following the inconclusive discussion, a firm proposal is required. (TJF)

Part II

Reports on the Sessions

Chapter 3

Review Sessions

3.1 Welcome and Introduction

Session Monday 3rd July – session 1

Chairman William Lupton

Programme Welcome and introduction to the Workshop.

WFL welcomed Workshop participants and summarised domestic arrangements. He then invited suggestions for modifications to the programme in the light of the open session on the previous Friday. The general consensus was that:

- more time was needed for discussion of new package features, with a specific session on error reporting.
- specific sessions were required to discuss command language features and the relationship between the command language and graphics applications.

WFL undertook to issue a new programme incorporating these changes.

3.2 Review of Progress on 1987 Workshop Recommendations

Session Monday 3rd July – session 2

Chairman Bernard McNally

Programme Review of previous workshop and of progress on implementing its recommendations.

ICL ICL is now in use everywhere except JAC; an estimate of one to two years for them to convert from ADAMCL was given. An ICL version of SMS has been released. It was noted that documents do not make it clear that ICL is now the preferred command language. A policy for the retirement of ADAMCL was required.

SMS There has been little development of SMS and hardware developments now mean that its future is uncertain. SCAR contains its own screen package which is a rival to SMS.

SCAR There is concern over ADC reliability and a fear that events will outdate the QCAR query language. The ADAMised version of SCAR was expected to be released very soon.

MON/NBS JAC still use MON (JCMT use an old version) - a policy on the retirement of MON was required. NBS is still not formally released in its own right but only as part of ICL.

Error Reporting Johan Hamaker had sent his RPT package to RAL for assessment but RAL had reservations about it and a more complete and portable solution was being sought. This would be discussed in more detail later in the Workshop.

ADAM Support In spite of the lack of effort available, some very useful enhancements had become available and bugs were getting fixed. The transfer, to Starlink, of responsibility for organisation and release of ADAM had improved that area considerably.

Telescope Interfaces The telescope control sub-committee proposed in Hawaii had not really worked because of the distances between members. However, JAB has designed VTI (Virtual Telescope Interface) as part of his work on the UKIRT telescope control system.

3.3 Reports from the Sites

Session Monday 3rd July – session 3

Chairman Keith Shortridge

Programme Reports by representatives of observatories, establishments, universities and Starlink on their use of ADAM, any extra rules that they adhere to, the way in which they use tasks, any non-standard components used, special techniques, problems.

This session took the form of a review of the way ADAM was used by the various represented establishments, the idea being to concentrate on the ways in which they had diverged from the 'standard' usage – whatever that might be. It was thought that this would help to indicate those areas in which standard ADAM was obviously felt to be deficient, assuming that people would only have diverged from it for good reasons, and that this might identify areas where ADAM could usefully be extended.

To give the discussion a target to aim at, KS presented a diagram of what he felt he understood by a 'standard ADAM system' (Figure 3.1). This diagram shows the message system running throughout the VAX, with assorted ADAM tasks hanging off it. D-tasks are connected to single instruments and have multiple actions concerned directly with those instruments, usually relatively complex actions that have to be rescheduled; A-tasks are not connected to anything, and perform one action, typically a data reduction or data assessment operation which does not need to be rescheduled; M-tasks (A-task monoliths) are collections of A-tasks linked into a single process to reduce the number of processes in the system; C-tasks exist to perform complex, possibly rescheduled, sequences of operations involving A- and D-tasks which are more easily programmed in such a task than as command language procedures. Any C- or D-task may have a noticeboard associated with it. A C-task is able to 'see' all A- and D-tasks on the system, in the sense that it is allowed to send them commands and to read from their noticeboards, whereas D-tasks are supposed only to be aware of the instrument they control, and are not supposed to initiate communication with other tasks. The command language, ICL, connected with the screen management system, SMS, is the only point through which the user communicates with

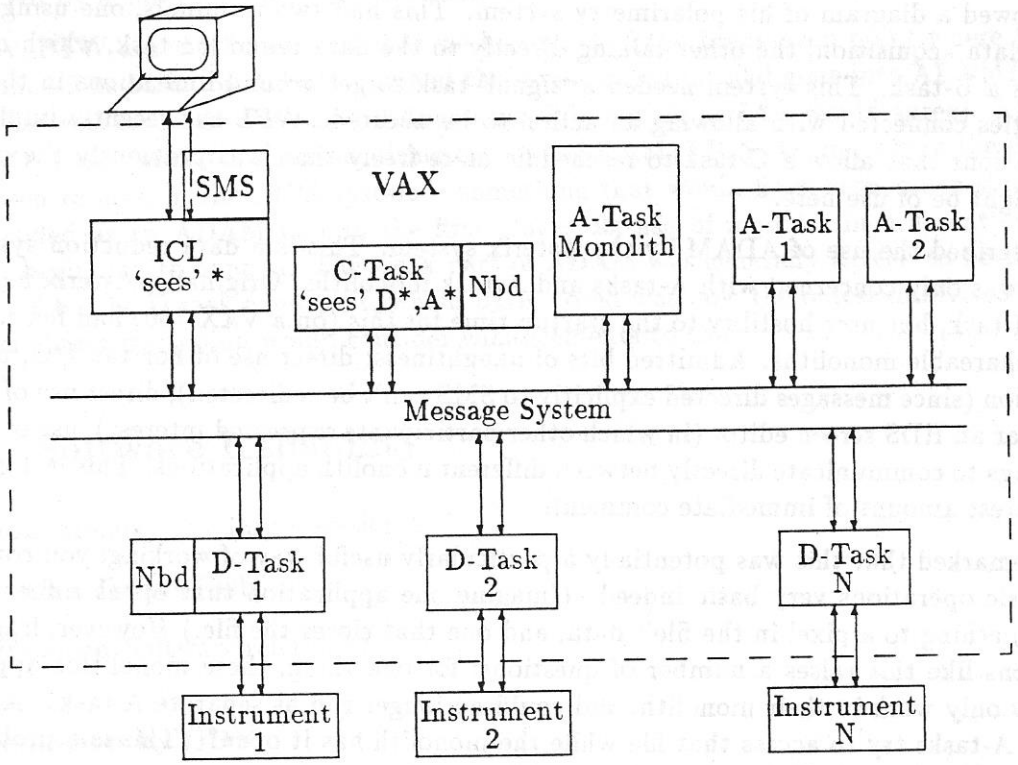

Figure 3.1: A 'Standard' ADAM System

the system. It can 'see' all the tasks on the system and control them directly, although it may use C-tasks to perform more complex sequences of operations.

Representatives of the various establishments were then invited to confess to their transgressions with regard to the rules embodied in this diagram, these sins to be marked according to originality and penances assessed.

PTW asked if you could have multiple copies of ICL, so that you could fire up multiple independent ICL procedures. Although this was not shown on KS's diagram, it was pointed out that SLW had always said the command language was just an application and you could have multiple copies of it. UKIRT are in fact using just that, and have multiple copies of ICL and SMS running together in observing configurations. The consensus was that this was perfectly OK, that a system could be configured deliberately in this way with co-operating copies of ICL, and that formal rules/locks to control who controls what were not needed.

The AAO diverge from the diagram shown by coding all their tasks as U-tasks, using JAB's U-task interface to let a task communicate directly with the observer. Although D-tasks are usually run with the U-task interface disabled, having a C-task run as a U-task allows a more direct feedback to the user than does the use of ICL/SMS, especially in the case of getting status information back to the user. Additionally, a change in the state of the user interface could be triggered by a hardware change. No strong opinions on this mode of operation were offered by non-AAO personnel.

The subject of task types being raised, WFL commented that rules about which task type to use in different circumstances are misleading, confusing and unnecessary. BDK added that having different fixed parts for all types of tasks was a mistake, but that moves were afoot to unify the

task types again.

JAB showed a diagram of his polarimetry system. This had two terminals, one using SMS to control data acquisition, the other talking directly to the data reduction task, which had been coded as a U-task. This system needed a 'signal' task to get around limitations in the C-task capabilities connected with allowing an action to be aborted. WFL has recently implemented modifications that allow a C-task to reschedule more freely than was previously the case, and these might be of use here.

RJV described the use of ADAM by the Asterix system. This is a data reduction system and as such was only concerned with A-tasks and A-task monoliths. Originally, Asterix had been a single M-task, but user hostility to the startup time for this (on a VAX 750) had led to the use of ICL shareable monoliths. Admitted bits of naughtiness: direct use of Fortran I/O, partly for redirection (since messages directed explicitly to SMS can't be redirected); direct use of SMG, in particular an HDS screen editor (in which other participants expressed interest); use of common block flags to communicate directly between different monolith applications. This last generated the greatest amount of immediate comment.

PTW remarked that this was potentially a particularly useful way of working; you could make your basic operations very basic indeed. (Imagine one application that opens a file, one that does something to a pixel in the file's data, and one that closes the file.) However, fragmenting operations like this raises a number of questions; for one thing, these monolithic applications will now only work in their monolith, and could no longer run as separate A-tasks. Also, what if other A-tasks try to access that file while the monolith has it open? (This is a problem that Asterix, with its single ICL process, does not have to face.)

The general feeling was that RJV had broken the rules by using common in monoliths, but that the resulting effect was highly desirable and ought to be supported. So, if the feeling of the Workshop is that this is a desirable feature, what are the rules for it? BDK pointed out that at telescopes, people are doing similar things, but are doing them in D-tasks (where the code for one action is allowed, even expected, to communicate directly with that for other actions). This discussion was continued at some length in later sessions.

An overview of the RGO use of ADAM was presented by LRJ. They are using or going to use a VAXStation 2000 as a user interface and MIMIC display (a graphical display of an observing configuration that changes in real time). At present this used VWS but would move to X Windows in future. They had a number of home-brew ADAM extensions: MIMIC and their data archive task used precompiled noticeboards; system changes to MIMIC were signalled by D-tasks firing ASTs in MIMIC itself (meaning that the D-tasks had to assume the existence of MIMIC, but MIMIC was a standard task used in all configurations); not all tasks in the system were in fact ADAM tasks – some D-tasks used sub-processes to handle asynchronous operations. The use of such sub-processes was found at other observatories, and were regarded by the Workshop as just a question of the internal implementation of the parent D-task; this was perfectly legitimate. RGO D-tasks were all written in Pascal, and there was a degree of standardisation of the interface to the instrument microprocessors through the RGO utility network that allowed new D-tasks to be produced relatively simply from a standard template. The other observatory representatives, with historically less uniform instrumentation, expressed envy. In a later session LRJ led a more extensive discussion on the question of AST usage.

DLT raised the question of shareable images and the importance of controlling what went into them, and WFL expressed the opinion that were it not for the possible overheads, the best system would be to have one shareable image per subroutine prefix. However, a number of establishments now have super-sub-systems such as PNB (at AAO) and PARMON (at UKIRT) which combine noticeboard and parameter system calls to simplify updating of parameters whose

values are also held in noticeboards for efficient access by other tasks. Shareable images were discussed by a sub-group later in the Workshop.

Finally, PTW asked if it was felt that we had reached the break-even point where the volume of applications based on ADAM justified the investment that had gone into ADAM? BVM said that that was certainly the case (although possibly not all at JAC would agree). KS said that at AAO it was definitely the case, citing the (non-ADAM) FIGS system, where the display task had been re-used in the IRPS system – something that would have been much simpler had it been coded as an ADAM task in the first place. Re-use of modules in different systems was finally beginning to happen. The weak part of ADAM was generally agreed to be the facilities provided by the D-task fixed part, which BDK noted was now the oldest unrevised part of the system. Later discussion would consider enhancements to this.

3.4 Software Reliability

Session Monday 3rd July – session 5

Chairman Dennis Kelly

Programme Software reliability.

Presentation by BDK

An interesting book discussing this topic is 'Software Reliability', by G J Myers. The following three definitions are taken from this book.

An ERROR is something which causes a piece of software to fail to do what a user could reasonably expect of it.

A FAILURE is the occurrence of a software error.

RELIABILITY is something like the probability of executing for a given period of time without failure weighted by the cost of each failure encountered.

These definitions are inevitably a bit fuzzy, but they give us a starting point for a discussion of reliability issues which might be important to us.

Consider a particular error in a piece of software where

F = frequency of failure
C = 'cost' of each failure
T = time before error is rectified

Total cost = F × C × T

So the difficulty of locating and fixing a bug is also important.

Now let us consider a pair of examples.

Say there is an application program called SPLOT which is in use on Starlink. Say there are 200 users of SPLOT and that each user invokes SPLOT 100 times per year. Also assume that SPLOT contains an error which results in failure once every 10^5 invocations on average.

runs per year = 20,000

Therefore a failure occurs once every five years.

In other words, SPLOT is unbelievably reliable.

Now consider the ADAM message system in use at the telescope. This is involved in every interaction between a user interface and a task, and in many interactions between tasks. During an observing session one may have two user interfaces plus a dozen tasks running.

One might have an average of one message per second.

This implies 36,000 messages in a 10-hour night.

A failure rate of one in 10^5 would mean a failure every three nights on average. Such a failure could lose 20 minutes observing time on each occasion.

This reliability is unacceptable.

Furthermore, this type of error is very difficult to track down. It could easily be several weeks or months before it was successfully identified.

The main question I have been gradually leading up to concerns the objectives of the ADAM support group. The question has been raised from time to time whether there is likely to be a disagreement between Starlink and the observatories over the priority to be attached to various aspects of ADAM support and development. In particular the subject of speed requirements has often been raised. I agree with PTW that this is not expected to be a problem area because Starlink is actively interested in speed problems. However, I believe there is a genuine difference between Starlink's requirements in the field of reliability and the requirements of the observatories.

Discussion

The point was made that the examples did not make a fair comparison. This was agreed, but it was also agreed that a valid underlying point was being made. The need for very high reliability in certain key system components was recognised. It was emphasised that the observatories had already undertaken work in this area, in that the ADAM V1 inter-task communication library had been subjected to code-reading sessions. Code reading had also been performed on the ADAM V2 MESSYS and MSP libraries and on the new D-task fixed part routines.

PTW drew the proposed organogram for the ADAM support group and pointed out that it specifically included support for real-time work. It was emphasised that achieving the level of reliability we were discussing involved identifying quality assurance as an explicit activity for the support group. It was also suggested that some involvement in code verification by the observatories might be desirable.

Chapter 4

ADAM Management

4.1 Documentation

Session Tuesday 4th July – session 1

Chairman Mike Lawden

Programme Documentation; following on from a discussion at RAL last October, Mike Lawden has produced a draft ADAM Guide. We should review the guide, identify gaps, and attempt to allocate people to fill those gaps.

Mike Lawden circulated a report on the current state of ADAM documentation and the third draft of his ADAM guide (SG/4).

He pointed out the problems with the current ADAM classified documentation (AED, AON, *etc.*). The fundamental problem is that it is not being maintained and has become out-of-date and inaccurate. In many cases it is also inadequate for the readers' needs. Dennis Kelly claimed that this documentation was of historical interest only, as it was written for a specific project which has come to an end. Unfortunately, this observation does not help the person searching for up-to-date information on ADAM – it merely explains why things are so bad.

It was agreed that the existing ADAM documentation should be merged with the Starlink documentation and that authors' home institutions should be acknowledged on the title page. Unfortunately, it wasn't made clear who is going to revise the existing documentation. This will have to be done by knowledgeable experts if it is to be of any value. This is the real problem which the re-organisation and merging recommendation does not address. I do not believe that the Workshop faced up to this problem. It was anticipated that the ADAM support group would generate some of the required documentation; in particular, data analysis and data acquisition programmers' manuals. Programmer-level documentation is required for all packages.

Many useful comments were made on the draft ADAM guide. More sections on application packages were suggested; in particular, ASTERIX (Bob Vallance), DAOPHOT (Nick Eaton), PHOTOM (SCAR), and SCAR (Helen Walker). Mike Lawden will produce a revised version of SG/4 for general release which will incorporate the new sections and suggestions.

4.2 ADAM Organisation and Release Mechanisms

Session Tuesday 4th July – session 2

Chairman Alan Chipperfield

Programme ADAM release mechanisms, bug/suggestion reporting mechanisms; are we happy?
Is there anything to be learned from what observatories are doing with release systems?
Would people read a VAX Notes conference?

The purpose of the session was to review the way in which ADAM is organised and released. Discussion took place under the following headings:

4.2.1 Release Mechanism

The Workshop expressed appreciation of Starlink's role in organising and implementing ADAM releases. The option of 'full' or 'mini' release was useful – many machines have only the mini-release installed to save on disk space. Partial updates, enabling fast updating via the network were also found to be useful. Because of more stringent system integrity requirements, Observatories tended to be some way behind Starlink in installing new releases.

4.2.2 Version Numbers

The current, rather haphazard system of version numbers was discussed. It was agreed that ADAMSTART should display an overall ADAM release version number (Action AJC). Proper use should also be made of the shareable image *major-id* and *minor-id* (Action AJC). LRJ stressed the need for upward compatibility between versions.

4.2.3 Release Notes

Although some complaints had been received, the Workshop agreed that the release notes were adequate. It was stressed, however, that release notes should not be used to document new features – relevant documents should be updated with the release.

4.2.4 System Testing

It was recognised that pre-release confidence tests should be more formal. Tests should include timings so that any efficiency degradation could be monitored (Action AJC, ASG). Observatories agreed to supply suitable tests for the control side of the system (Action Observatories).

4.2.5 Directory Structure

The directory structure was reviewed. There were no real problems with it but JAC and LPO reported that they keep multiple versions of ADAM online (as indeed do RAL) and this should be borne in mind. In particular, a single run-time directory might be helpful in switching between versions.

The question of how best to include stand-alone Starlink software items into the ADAM shared images was discussed at a splinter session.

AJC asked the Workshop members to be alert for items which could be removed from the release because they were no longer generally used (Action All). Items identified were: MON, ADAMCL, DIAGRAM, ENGIF and CAMAC. It was agreed that a warning would be given before items were removed.

4.2.6 INCLUDE Files

It was agreed that standardisation of INCLUDE file names would be helpful but standards for all languages would have to be defined. There was no support for keeping INCLUDE files in libraries.

4.2.7 CMS/MMS

The suggestion that CMS should be used for code organisation was viewed favourably, particularly with the prospect of a larger group working actively on the system. Reports of MMS for system building were less favourable and alternatives should be considered.

4.2.8 Bug Reporting

All agreed to report bugs via RLVAD::STAR. Support programmers should also forward, to RLVAD::STAR, any bug reports sent directly to them (Action All). Starlink is about to set up a new bug reporting system including a VAX Notes conference which may be read by anyone wanting to know what bugs have been reported.

4.2.9 Proposal Submission

The ADAMSC route for proposal submission could continue but it was agreed that a VAX Notes conference for discussion of system developments would be useful. AJC agreed to set up and monitor such a conference (Action AJC).

4.3 Splinter Session on Shareable Image Organisation

Session Wednesday 5th July – splinter session

Chairman Alan Chipperfield

Programme To discuss Starlink's use of shareable images.

Two main questions were addressed:

1. Whether or not ADAM should be de-coupled from releases of other stand-alone Starlink software items.
2. Efficiency aspects of shareable images.

4.3.1 De-coupling

Although it was recognised that the use of shareable images containing stand-alone Starlink software items, such as GKS, could cause difficulties for observatories where updates have to be carefully controlled to maintain system integrity, the benefits in automatic inclusion of latest updates outweighed this. The use of logical names enables switching back to old versions. It was therefore agreed that ADAM should, where possible, incorporate Starlink stand-alone shared images. DLT would review Starlink's use of shared images with this in mind. (Action DLT) This was recognised as a fairly long-term goal which would result in a release which was not upwardly compatible.

4.3.2 Efficiency

It was clear that the effect on image load time of having multiple shared images was unknown and that more tests should be done. Tests at ROE had shown that installing shareable images gave a slight time advantage.

Chapter 5

Package Management

5.1 Error Reporting

Session Tuesday 4th July – session 3

Chairman Bob Vallance

Programme Error reporting.

This session sought to discuss the problem of low-level error reporting and to find a solution to the '%HDS-I-FILNF, file not found' syndrome (about which considerable concern was expressed at the open meeting the previous week). This problem comes about because of the layering of ADAM software and in particular because the ERR/MSG systems sit above the parameter system. At the point where the error message is constructed, all knowledge of the original context of the error is lost, so the system cannot report, for example, which file has not been found.

There had also been discussion of the same problem at the 1987 Workshop (Proceedings, p8–10) and a recommendation to produce a solution (p1–2). A tentative interface definition was formed at that workshop and it was reported that Johan Hamaker (JCMT) has since produced the RPT system to comply with it. The Starlink representatives at this workshop confirmed that they had received the RPT system from Johan but that it hadn't been implemented in Starlink software or released for general use because they had some reservations about it. Rodney Warren-Smith raised the following concerns:

1. It was generally too VAX specific.
2. In particular it relied on the VAX message system.
3. Subroutines had only two arguments (no parameter name) and therefore that the functionality of the existing ERR system could not be implemented using it.
4. Communication via STATUS is undesirable as systems outside ADAM (*e.g.* GKS) don't use it.
5. Other small implementation details.

Bernard McNally reported that the system was already in use at JCMT as objections had not been raised earlier. There then followed some general discussions about what RPT did and what

was desirable. No definite conclusions were reached and Rodney was actioned to to formulate a proposal for changes to RPT.

Actions:

1. Make proposal for changes to RPT – Rodney Warren-Smith.
2. Make code and documentation for RPT available – Dave Terrett. (This was done after the session.)

5.2 Package Problems 1

Session Tuesday 4th July – session 4

Chairman Patrick Wallace

Programme Package problems, wishes, plans; a large area including D-task routines, the parameter system, HDS, NBS; support for multiple languages; whatever people want to discuss.

Discussion centred around the parameter system and the provision of extra facilities.

5.2.1 Parameter Help

It was agreed that the single line of help, available when '?' is typed in response to a parameter prompt, is too limiting and that hierarchical help should be used to provide multiple lines. We should use VMS help. JAB agreed to write a detailed proposal.

5.2.2 Keyword Abbreviations

It was agreed that there ought to be abbreviations for keywords (including reserved keywords) or automatic minimum abbreviation determination; maybe both? The automatic method was greatly favoured. AJC to investigate.

5.2.3 MIN and MAX

It was agreed that one should be able to specify a minimum and maximum value for a parameter which would be taken if 'MIN' or 'MAX' was typed in response to a parameter prompt. WFL to make proposals.

5.2.4 ACCEPT on Prompts

It was agreed that '\' should be accepted in response to a prompt to cause all unset parameters to take their prompt values. The method was agreed and AJC would implement it.

5.3 Package Problems 2

Session Tuesday 4th July – session 5

Chairman Rodney Warren-Smith

Programme Continuation of package discussion.

5.3.1 Parameter Checking and Conversion

NRH gave a presentation of the RGO's system for control of the WHT, based on the use of control files. He outlined the way the system works and the role which ADAM tasks played within it. He and LRJ then discussed some of the problems which had been encountered.

Since the system is 'data driven', it was felt necessary to allow D-task parameters to be tested (*e.g.* for validity/conflicts) or converted (*e.g.* between different units/data types) in a variety of ways before the task action began. This was not always possible at present. A example cited was that _INTEGER filter positions which the user had (erroneously) specified as real numbers were being automatically converted to integers when read by the task, so that their validity could not later be checked.

It was argued that this type of checking and conversion might be accomplished through additions to the interface file specification. The consensus, however, was that the range of possible requirements in this area was too large (and potentially specialist in nature) to contemplate making them all available in a general purpose system (ADAM). PTW also remarked that highly complex 'control-file driven' systems which depended on specialist file parsers had a habit of being abandoned once their implementors had left, and that it was generally best to obtain flexibility through the direct use of Fortran whenever possible.

It was noted that the RGO's filter number problem could be circumvented by changing the parameter type to _REAL, and performing the conversion and checking within the task.

A possible general solution might be to allow a section of user-written code to be called from the task fixed part prior to invoking the main body of the task. This code could then perform parameter checking and conversion in an arbitrarily sophisticated manner. For this to work, parameter system enquiry routines would be required so that the interface file contents could be acquired. WFL noted that a number of such routines already existed and could be made available for this purpose.

LRJ would produce a proposal for adding a D-task checking and conversion routine to ADAM, and would draw up a list of the parameter system enquiry routines which would be needed.

5.3.2 Strict Typing of Interface File Entries

It was thought unnecessary to restrict the data type conversions permitted on parameter values typed in by users; *i.e.* floating point numbers should continue to be converted to integers if required, rather than re-prompting. However, data typing within interface file entries should be made strict, so that, for instance, the RANGE of an _INTEGER parameter could only be specified using integers and not floating point numbers. An error should result during interface file compilation if data type mis-matching was detected.

5.3.3 Case Conversion

The discussion turned to whether it would be useful to perform automatic conversion to upper case when character (or literal) parameters values were acquired. This would remove the case conversion burden from the programmer, but might not always be required. It was agreed that an optional CASECONVERT keyword in the interface file should be implemented for such parameters, *e.g.*:

```
CASECONVERT UPPER    # Converts to upper case automatically
CASECONVERT LOWER    # Converts to lower case automatically
CASECONVERT NONE     # (The default) no case conversion
```

5.3.4 Parameter System Efficiency

RJV commented that the parameter system often seemed slow, implying inefficiency, when called upon to do trivial operations which should require little processing. However, BDK pointed out that this subjective assessment might be incorrect and that time could be lost in context switching (the 'un-greed' effect) when running from ICL. This was beyond the parameter system's control and was likely to depend greatly on machine loading. The performance from DCL might be better, although it was also possible that HDS was causing the problem.

It was agreed that there was little point in discussing performance without specific examples and timing tests to identify the bottlenecks. RJV would perform some tests and compare ICL and DCL behaviour on specific examples and report the results.

5.3.5 Changes to Parameter System Behaviour

Three of the proposals outlined in AJC's paper (ref. Appendix D), which were designed to eliminate anomalies in the present parameter system behaviour, were agreed after brief discussion, namely:

1. In the CANCELLED state, a parameter's VPATH should always be ignored and a prompt forced.
2. In the RESET state, the 'current' parameter value should be ignored on both the PPATH and the VPATH, rather than on the PPATH alone as at present.
3. Names should be entered into the parameter system storage as soon as they are obtained, in the same way as primitive values, rather than waiting until the task ends, as at present.

A further proposal, to implement a routine to set a parameter into a specified state explicitly, was designed to overcome a problem which MJC had noted where a parameter needed to obtain a new dynamic default (using VPATH 'DYNAMIC') several times, such as in a loop. In this case, the parameter was cancelled to remove the previous value, but this resulted in the user being prompted instead of simply obtaining the new default value. The proposal was to allow the parameter to be set into any state, so that a new value could be obtained using the original VPATH.

As well as the problems noted in AJC's paper, the proposed new PAR_ routine would require the user to have access to an INCLUDE file defining global constants at a lower level (SUBPAR).

This was felt to be an undesirable anomaly which eroded the insulating PAR subroutine layer by making its internal mechanisms visible to the user. In addition, a number of other subroutine packages layered upon the PAR system would need to change in order to offer similar facilities.

AJC would investigate alternative solutions to this problem.

Chapter 6

Programming

6.1 Tasking Architectures

Session Wednesday 5th July – session 1

Chairman Tony Farrell

Programme Unification of ADAM task types.
Retention of context between action invocations.
Delivery of AST's to other processes.

6.1.1 Unification of ADAM Tasks

TJF displayed overheads showing the differences between ADAM task types.

Task Type	Abilities
A	P,RT
M	P,RT,MP
D	MN,R,CN
CD	MN,R,CN,R,RT
CM	MP
U	P,MN,R,CN,U

Task Abilities:

P	Prompting
RT	Parameter system reset between action entries (invocations)
MP	Multiple actions each with parameters local to each action
MN	Multiple actions, access to all parameters, needs lists
R	Rescheduling of actions
CN	If checking of the needs list done
U	Status display/Command interface

MP (monoliths) and MN are mutually exclusive

It was agreed the current system was restrictive and confusing.

TJF suggested there are three basic programming styles for ADAM tasks and that this could form the basis of an improved system. There would be link procedures for each of

A A-Task style (one action)
M Monoliths
R Rescheduling

With modifiers in the interface file to enable or disable the other features, *e.g.*:

```
INTERFACE PROGRAM
   PROMPTING ENABLED
   PARSYS    NORESET
   UTASK

         etc...
```

And for the needs list:

```
ACTION RESET
   OBEY
      CHECK_NEEDS ENABLED
      NEEDS PAR1
   ENDOBEY
ENDACTION
```

WFL suggested modifications he recently proposed to the D-task fixed part could allow the system to use only one task type. These proposed changes were examined.

Scope of the changes:

- Tasks can control multiple actions in other tasks whilst rescheduling.
- DEVINIT need not be supplied.
- The programmer can provide an 'A-task-like' main application routine and GET/PUT arguments using a set of TASK_GET_* and TASK_PUT_* routines. (This is the particular item which allows only one task type to be used.)
- D-tasks can prompt and use MSG/ERR routines. A full set of encode/decode routines for passing command lines to actions and receiving values back from actions is supplied.
- D-task routines are put into the shareable image (all except MAINTASK).
- A new DLINK supports both old and new styles of programming.

The changes above allow the programmer of a rescheduling task (*e.g.* a D-task) to provide an 'A-task-like' main application routine; *i.e.* a main routine with only the status argument. To determine the action name, he would call TASK_GET_NAME. Likewise, the main routine of a monolith would also call TASK_GET_NAME before determining which application to call. A-tasks would not change.

Other features would be determined by interface file flags, as per TJF's suggestion.

The Workshop accepted WFL's changes as part of the solution. The following items were noted:

- The use of the TASK_GET_* functions by application tasks should be discouraged or prohibited.
- Details of shareable images mentioned in WFL's notes are to be discussed at a further meeting.
- Details of the tidying up of the D-task fixed part are to be discussed at a further meeting.

Details of the changes to the interface files are to be determined at a later date, as part of a proposal by TJF.

Upwards compatability is to be maintained where possible.

Conclusion: The Workshop agreed in principle to simplify tasking architectures along the above lines. TJF is to submit a detailed proposal. WFL is to complete and release his system modifications.

6.1.2 Retention of Context Between Action Invocations

Much discussion centred on how context should be maintained between action invocations. For example, at present each action in a monolith will reset the parameter system on exit (done by the fixed part). As part of this, HDS locators derived from parameters will be annulled. This is done because each action in a monolith is considered to be a separate application, each of which can be implemented in a separate A-task, if required.

In fact, because the monolith is a single process, and due to the way HDS works, there is some persistence between action entries. Some programmers have been illicitly taking advantage of this to improve performance.

It was pointed out that the cost of releasing all resources, such as HDS and graphics devices, at the end of an action and re-allocating them for the next action was high.

It was considered desirable, for performance reasons, to enable a task to maintain resources across actions whenever possible. A long discussion ensued. The major problem in not releasing resources when an action completes is ensuring a resource is released before another task wishes to use that resource. Various techniques for implementing this feature were examined – resource lock managers and using ICL, for example.

The following method was devised:

- There will be various resources managed by the command language; currently only 'CONTEXT' is defined.
- The interface file will list the resources used by the task. When a task is loaded, ICL will obtain a list of the resources used by that task.
- ICL will maintain an internal list of all possible resources. The name of the last cached task, to which an obey was sent, that requires that resource is held in this list. The name of the last cached task to which an obey was sent is also stored.
- Each time an obey is sent to a cached task which is different from the last cached task, ICL does the following:

It examines the list of resources required by the task and the internal list of resources. If the new task is different from the one which last required that resource, then an obey is sent to the task mentioned in the resources list, specifying an action used to release that resource. The name of the new task is then put in the list.

It was suggested the actions needed to implement this system and any associated parameters could be set up by the D-task fixed part. They would therefore not be required in the IFL file.

There was further discussion on this topic in session 2, on Friday (ref. Section 6.3).

In this supplementary discussion, the following points were noted.

- If we use CONTEXT now, intending to change to specific resource names later, the implementation could be harder. We could only identify HDS and GRAPHICS as possible resources and it was suggested we start with these.
- Instead of modifying the IFL, we could use the define command in ICL to define resources and to release resources.
- In this system, old programs are considered to grab and release all resources at every action invocation, hence old programs can co-exist with new ones.
- Is is worth having a method of allocating and releasing files and devices? Much discussion centred on this point. This could be very hard to implement and for this reason the idea was discarded.
- PTW suggested we only have CONTEXT because it is simpler and he thought it would cope with 95% of cases. Others thought we would lose too much by doing this.

Conclusion: The Workshop agreed in principle to support more detailed task context control. TJF is to submit a definite proposal.

6.1.3 Delivery of ASTs to Other Processes.

LRJ circulated a note describing problems with the AZ$SNDAST system service. In particular, if the receiving task did not set up the AST or set it up incorrectly, it could fail with an access violation when the AST is received.

LRJ suggested a solution in his note but this seemed to require a structure to be set up in a known address (LIB$COMMON or the process header). This solution was considered to be a bit dirty.

Another suggested solution was that some sort of checksum be set up in memory near the AST address (or a data structure containing the AST address). The checksum could be used on AST reception to ensure the task had enabled the AST. The major problem here is that the checking must be done in a high access mode, otherwise, the checking code could also fail with an access violation. LRJ suggested the technique of piggyback kernel mode ASTs, used by VMS in some system services, could be used.

Conclusion: It was agreed a problem existed. TJF and LRJ are to discuss the problem with Lewis Waller.

6.2 Splinter Session on D-task Fixed Part

Session Thursday 6th July – splinter session

Chairman Jonathan Burch

Programme To discuss proposals for D-task fixed part enhancements; particularly those raised in the document 'Time to Raise Cain' (ref. Appendix E).

The meeting decided that an enhanced DTASK_ASTSIGNAL, which can be called any number of times in main line code and at AST level, is required. (BDK) A facility for delivering a user AST to a control task on receipt of a message from a subordinate task was also discussed, but decided against. William Lupton's ACT__MESSAGE scheme goes some way towards satisfying the needs that the AST facility would have been aimed at.

The need for concurrent and/or queued task actions was discussed but it was not clear that they are worthwhile. A detailed proposal needs to be produced. (JMB) The suggestion that a facility for invoking one action from another in the same task should be provided, was also discussed but it was decided that such a facility is not necessary.

The meeting agreed that it is desirable for the fixed part to deal with incoming commands asynchronously, allowing them to be serviced immediately whatever the task is doing. This would solve the problem of command requests being held up if the task's ACT routine is active, but would cause implementation difficulties. A 'feasibility study' needs to be carried out. (JMB)

6.3 Miscellaneous Discussion

Session Friday 7th July – session 2

Chairman Tony Farrell

Programme Miscellaneous; general discussion on new hardware and software possibilities.

In the event this session became a discussion on various stocking filler items and further discussion on context retention developed. Items discussed were:

- Interrupting ADAM tasks.
- Setting the default directory.
- A new context for reserved action names.
- Retention of context between action invocations (see Section 6.1.2)

6.3.1 Interrupting ADAM Tasks

Currently, while a task is executing applications code, there is no method for aborting that code and forcing a return to the fixed part so that messages can be received. For example, if, while executing an OBEYW from ICL, the user types Ctrl-C, ICL aborts waiting for the OBEYW. However, the task to which the obey was sent continues to execute. This is not what the user expects.

The Workshop considered it desirable that a system be implemented to enable an ADAM task to be interrupted while it is executing the applications code. An implementation based on the command language delivering an AST to the ADAM task was agreed to.

It was suggested that the user interface interrupt cached tasks if an OBEYW is in progress to that task, and Ctrl-C is typed. For uncached tasks, ICL can provide a command which will interrupt a task.

There is a question of whether any task (either cached or uncached) for which an OBEYW is in progress, should be interrupted on receipt of Ctrl-C. Cached tasks are normally controlled by OBEYWs and thus would normally be interrupted. Uncached tasks are not always controlled by OBEYW and, in the instances where they are not, it is not desirable that they always be interrupted. It was pointed out that uncached tasks may be controlled from different locations and therefore should be interrupted with a specific command.

ICL's ability to call an exception handler on reception of Ctrl-C could be used, by programmers of control systems, to determine if uncached tasks should be interrupted. Unfortunately, ICL cannot, at present, provide an outer level exception handler.

It was suggested that the AST routine in the task receiving the AST should set a flag to indicate an interrupt has been received. The applications code could then call a routine to check if the flag has been set.

The following routine call was suggested:

> `TASK_BREAK(STATUS)`
>
> If status not zero on entry and if an interrupt has occurred, clear the flag, set status and report using ERR_REP. If no interrupt has occurred since entry to ACT, or the last call to TASK_BREAK, then do nothing.

The flag must always be cleared on entry to ACT to ensure that interrupts received before an action is invoked are ignored.

The use of such a routine is a very clean way of implementing an interrupt system, but it was pointed out the applications code must check the flag regularly. Sometimes this is not possible or practical.

A more complex implementation would allow the above system to work, but enable the application programmer other options, such as calling a condition handler to unwind the stack. This is potentially dangerous since, for example, some Fortran RTL routines cannot be unwound and then re-activated successfully. But used with care, unwinding the stack can be used and the ability to call a condition handler on reception of an interrupt can be very useful. Also, there is no need to check the flag.

In this implementation, the AST routine, called on reception of an interrupt, would call LIB$-SIGNAL to force the process out of AST mode and activate a condition handler.

In this system, the D-task fixed part will have set up a condition handler on startup and another on entry to ACT. The first one simply dismisses the signal (it is called while the process is in the fixed part). The second one will set the flag used by TASK_BREAK, before dismissing the signal. (The flag is cleared before the second handler is set up.) At this point, the system operates as per the simple implementation.

The user's code may then establish its own condition handler to catch the signal and do what ever it wishes (unwind the stack, continue, *etc.*).

It was pointed out that this method may not be portable. TJF believed it could be made so, as some or all versions of Unix support a signalling system and a function called 'longjmp'. These may be able to be used to implement such a system under Unix.

Conclusion: The ability to interrupt ADAM tasks is desirable. TJF is to submit a definite proposal based on at least a method involving checking of a flag by applications code. He is also to investigate if an implementation allowing the use of condition handlers can be implemented in a portable manner.

6.3.2 Setting the Default Directory

Currently, ICL provides a method of setting the default directory in its own process and any DCL process it has created. Unfortunately, it has no way of setting the default directory in any ADAM task it may control.

It was agreed a method of setting the default directory of ADAM tasks was required.

Directory defaults of *cached* tasks would be set by the ICL command, DEFAULT. Directory defaults of *uncached* tasks should be set by a specific command since an uncached task may be controlled from several sources. It is also desirable for the applications code to return to the fixed part before the directory is changed, to avoid it being changed in the middle of an action.

The following basic method was suggested.

> An action, 'DEFAULT', and associated parameter are defined by the fixed part. When this action is received by the fixed part it sets the new default directory to the value specified in the parameter.

The Workshop deemed it desirable for the programmer to have some way of overriding this default action. It was suggested the default action could be overridden by an action of the same name being defined in the interface file. In this case, application code could be called to set, or ignore, the action.

Conclusion: The ability to set the default directory is required. TJF is to submit a proposal.

6.3.3 A New Context for Reserved Action Names

In the course of discussions on setting the default directory, it was pointed out that we now have two instances requiring reserved action names. Both the maintenance of resources and setting the default directory require them. It is not hard to foresee requirements for more reserved action names – initialise, exit *etc.*

The Workshop noted a problem in the maintenance of upward compatibility when we start defining reserved action names. For example, many people already use the action name EXIT.

It was suggested, and strongly supported, that we define a new message context for this propose. (Currently, message context must be one of SET, GET, OBEY, CANCEL.)

There was some dissent to the idea along two lines. It was suggested there could be a lot of work involved. It was hard to see why this would be so. It was also pointed out that monoliths do not currently require the CONTEXT argument used by D-tasks, but could do so if a new context was introduced. This is not seen as major problem, as they could use the TASK_GET_CONTEXT call to be introduced by WFL's changes to the D-task fixed part. (Note: the D-task fixed part is used in all task types.)

Conclusion: TJF is to investigate and submit a proposal.

6.4 Command Languages

Session Wednesday 5th July – session 2

Chairman Peter Allan

Programme Command language features.

Although the title of this section was 'Command Languages', it quickly became apparent that in practice this meant ICL. Several points were brought up in the Open Meeting for discussion in the Workshop. These were

- Precision.
- Log files.
- Lexical functions.
- Syntax
- Array handling.
- Access to HDS.

6.4.1 Precision

It had been suggested in the open meeting that ICL variables should use double precision rather than normal reals for standard floating point variables. Peter Allan gave a résumé of the accuracy that could be achieved using double precision and it was agreed that it would give sufficient accuracy for positional data for any foreseeable future developments. (Clive Page felt that even double precision would not be sufficiently accurate for timing data in some circumstances, and gave the example of time tagging of photons to nanosecond accuracy while storing the values as Modified Julian Dates.)

As it was an easy change to implement, and provided most of the functionality required, the Workshop recommended that ICL should be modified to use double precision for floating point scalar variables.

6.4.2 Log files

ICL already has a log file to record a session, but it was felt that it would be useful to have a facility whereby one could recall the previous N commands into a file for subsequent editing. This would allow one to type in an ICL procedure directly, to make an error and then to recall the typed commands into a file so as to correct the error. It was felt that this would be a useful facility, and furthermore that it would be easy to implement as it could be done using the same method as that used for writing the current log file.

6.4.3 Lexical Functions

A requirement had been expressed to have various VMS lexical functions (F$ELEMENT in particular) included in ICL. While it was agreed that this would be very useful, Peter Allan suggested that it should be done with due regard to portability. Jeremy Bailey said he would provide a few of the simplest lexical functions in a future release of ICL.

6.4.4 Syntax

Lewis Jones suggested that the syntax of ICL should be modified to be closer to that of DCL, in particular, it should allow command qualifiers, *e.g.* TYPE/PAGE. The argument in favour of this was that general users found this style of command interface easy to use. However, it was felt that it would be undesirable to base ICL too closely on DCL as this could cause confusion. On a VMS system, users could become confused as to what was DCL and what was ICL, leading to unreasonable expectations as to what would be provided on a Unix system.

It was suggested that the ICL command 'DCL' should be replaced by a dollar as this command really means 'use the computer's command language' rather than 'use DCL'. On a non-VMS machine, the two things are not the same. In a similar vein, it was felt that the ICL command 'DIR' should be removed as it was confusing that this was the only DCL command available directly from ICL. With the replacement of the 'DCL' command by $, this would mean that a user would have to type '$ DIR' instead of 'DIR'. JAB said that he would put these changes into a future release of ICL.

6.4.5 Arrays and Access to HDS

A requirement had been expressed to have arrays available within ICL. This was agreed to be a good idea, but it was not clear how to provide this facility. There was the question of whether floating point arrays should be stored as reals or as double precision. Also the way the arrays would appear to the user was unclear. It was felt that it would be very useful to have the arrays implemented as HDS structures in GLOBAL.SDF (or an equivalent) so that the arrays that had been manipulated in ICL would be directly available to data reduction tasks. The fact that ICL variables are not implemented like this means that you cannot write to an ICL variable from a program if the variable is not given on the command line. It was felt that it would be very useful to be able to do this. Figaro has a definite requirement to do just this and KS said that he would implement Figaro variables as elements of GLOBAL.SDF for the ADAM monolith version of Figaro. The only objection to the use of GLOBAL.SDF is that it was felt it would be slower than the current implementation of ICL variables.

There was a general requirement to be able to manipulate HDS structures and Bob Vallance said that Asterix 88, which was about to be released, would contain an HDS editor. The first version would probably be a read-only version, but future versions would be full editors.

6.5 Graphics and IDI

Session Wednesday 5th July – session 3

Chairman Dave Terrett

Programme Graphics and IDI

The ADAM Open Meeting identified the need for some general command-driven 2D graphics in ADAM. Two existing software systems were referred to in the discussion:

- MONGO – a stand-alone program which can read tables of numbers from text files and plot simple X,Y graphs.
- IDL – which has commands for plotting command language arrays.

Three approaches to providing similar functionality in ADAM were discussed:

1. Build graphics commands into ICL.
2. Write a MONGO-like ADAM task.
3. Write a MONGO-like 'DEFUSER' image that can be called by ICL.

Option 1 was rejected because there was no prospect of any significant further development of ICL until the ADAM support group had been set up and work could not even start before ICL arrays had been implemented.

Option 2 would limit the usefulness of low level drawing commands (*e.g.* draw-line, plot-point *etc.*) because of the overheads in inter-process communication and context switching, and so it was agreed that option 3 should be adopted.

It was also agreed that such a package should be based on PGPLOT rather than MONGO because of the higher standard (of both efficiency and maintainability) of the PGPLOT code and because PGPLOT was already a supported part of ADAM. Two MONGO-like packages based on PGPLOT have already been written (by Paul Harrison, Jodrell Bank and Clive Page, Leicester). Clive Page undertook to distribute details of his system to the Workshop participants.

Chapter 7

New Features

7.1 Progress on New Software

Session Thursday 6th July – session 1

Chairman Dennis Kelly

Programme ADAM version 2, Figaro and ADAM, SCAR and ADAM: what is happening on these fronts and what problems remain.

7.1.1 SCAR

DLG pointed out that he had experienced systems competitive with SCAR which were much faster. He thought that the way the catalogues were stored on disk was relevant and that he intended to investigate this area further. It was emphasised that a series of reports on the speed of SCAR had been produced in the past by Clive Davenhall. These showed that SCAR was not I/O bound.

In discussion it was decided that the speed of SCAR should be investigated, the specification of the ADC subroutine interface should be compared with other systems, and in conjunction with this the ADAM support group should be asked to investigate supporting ADC as part of the ADAM system.

7.1.2 Figaro

KS described the current state of the ADAM Figaro monolith. This existed, and was available for trial by members of the Workshop, but he felt that it was not yet quite ready for general release. He was worried about the following aspects:

- At present there was no on-line help for parameters. The existing multi-line Figaro help could not be converted automatically to the single line allowed by ADAM.
- 'MIN' and 'MAX' as responses to numeric parameters were not supported by the ADAM parameter system and Figaro users had got used to having them available.
- The parameter system did not allow abbreviations of parameter and keyword names. The Figaro system did, and the names had been chosen on that assumption, so using the full names was awkward, and the minimum abbreviations (which the current version of the monolith used) were neither obvious nor known to most Figaro users.

- GNS names were not supported properly by the system.
- There was no mechanism for National, Local and User level extensions.
- It was unclear how the 'Figaro User Variables' should be stored. The current system worked unchanged under ADAM, but it was not possible for the values to be obtained and made use of by ICL procedures.
- The system was essentially undocumented.

However, despite these problems, most of which were parameter system limitations which the Workshop was already considering, he felt enthusiastic about the system. It was the fastest available way of running Figaro applications, and allowed Figaro to be programmed using ICL rather than DCL.

In the following discussion it was accepted that the parameter system limitations would be addressed anyway and that the GNS problem was genuinely trivial (indeed, it was fixed during the Workshop). The user variables would best be handled by making them ADAM global parameters, which ICL could access through the 'GETGLOBAL' command. There was a general desire to get the Figaro monolith available to the user community as soon as possible, but this was tempered by wanting to ensure that they would not be deterred by the existing limitations. Eventually it was decided to aim at having a first release of the system in mid-October[1]. This release would have the GNS and user variable problems sorted out, would have some basic documentation and would produce some limited on-line help, making use of ICL's 'DEFHELP' facility.

7.1.3 ADAM V2

DLT expressed strong concerns about the security implications of enabling the ADAM V2 networking ability on Starlink machines. As enabling networking is optional, this need not delay the release of V2, but DLT and BVM were actioned to investigate how the networking could be rendered secure.

7.2 Workstations

Session Thursday 6th July – session 2

Chairman Jeremy Bailey

Programme Workstations; VMS and Unix ones. How are people using them and how do they fit into ADAM?

JAB described the use of a VAXstation 3200 for the new UKIRT telescope control system. On this system DECwindows application programs are used to provide the user interface to a telescope control system kernel. An ADAM D-task is also used to allow access to the system from the instrumentation computer via the network ADAM message system.

Most of the discussion centred around the use of X Windows and DECwindows and their role in future ADAM systems. It was generally agreed that user interfaces based on these systems were a good thing and would provide a considerable improvement on the existing SMS system

[1]The ADAM Figaro monolith is now due for release in mid-January 1990, Ed.

used at some observatories. The role of such systems in data reduction was less clear. Starlink cannot currently afford workstations for all its users. Lower cost 'X terminals' may help to make this type of user interface more widely available, but would probably also require an increase in CPU power to support many users running DECwindows.

It was decided that the University of Edinburgh should apply for a grant for a programmer to work on an X Windows based user interface for ADAM which could replace SMS.

7.3 Telescope and Instrument Interfaces

Session Thursday 6th July – session 3

Chairman Jeremy Bailey

Programme Telescope and instrument interfaces.

Two questions were considered under this heading.

1. Should we attempt to standardise the interface to instrument and telescope D-tasks?

   The aim of such standardisation would be to simplify the porting of instruments between telescopes. A standard interface to the telescope would enable instrument software to be more easily moved from one telescope to another. The previous ADAM workshop had recommended the setting up of a working group to specify a standard telescope interface. Extending such standardisation to instrument D-tasks would simplify cases where, for example, an instrument is used with different detectors at different telescopes. (The Hatfield polarimeter is used with the IRPS at the AAT, and with the UKT systems at UKIRT.)

   The individual observatories have generally adopted some level of standardisation of D-task interfaces. Some examples are the CDI interface used for the heterodyne receivers at the JCMT, and the generic detector D-task used for the CCDs and IPCS at the AAT. However, there has been no standardisation across observatories. The general feeling of the Workshop was that such standardisation was not practicable in the case of instruments, because of the wide range of different instrument capabilities at the different observatories. Standardisation of the telescope interface was still thought to be worthwhile.

2. Should we standardise the interface between D-tasks and instrument control microprocessors?

   Most instruments controlled by ADAM systems have their own microprocessors performing the low level control functions. At present the communication between the VAX and the microprocessors lies outside the scope of the ADAM system. Now that the ADAM message system has been extended to operate over network connections between VAX machines, it is in principle possible to extend the ADAM message system to operate over the links between a VAX, and another machine such as a microprocessor system. BDK is investigating such an approach for communications between the VAX and a transputer system for the SCUBA instrument on the JCMT. The approach could be extended to other systems such as the utility network being used on the WHT. The Workshop concluded that standardisation of these interfaces was worth investigating.

7.4 ADAM Portability

Session Thursday 6th July – session 4

Chairman Dave Terrett

Programme System portability; the problems of moving the ADAM system itself to run under different operating systems; scope for simplification and rationalisation.

Pat Wallace outlined the steps that Starlink has taken to allow ADAM data reduction tasks to be run on a Unix system: A DECstation 2100 is being ordered for RAL and some part-time consultancy is being purchased to assist in the setting up of the system. An attempt will then be made to port a limited subset of the ADAM environment so that at least some data reduction tasks can be run; probably directly from the Unix shell (*cf.* running ADAM tasks from DCL). This exercise would probably exclude ICL, SMS and all inter-task communication. It was noted that the target system would initially have to be Ultrix on the DECstation and not, at this stage, any sort of generic Unix.

The two most important aims of the port will be to gain enough experience to be able to manage a contract with a University for the port of the full system and to discover what rules applications code need to adhere to in order to run both on Unix and VMS ADAM.

The observatories were asked for their views on the desirability of porting ADAM.

- The AAO and ROE expressed the view that it was essential (for data reduction at least). For data acquisition it was less important because the price/performance of a system for data acquisition was not usually an issue.
- RGO were concerned that porting ADAM might divert support away from the VMS version and stated that any reduction in the efficiency of the VMS system in order to facilitate the port would be unacceptable.

It was agreed that a port should not be achieved by reducing the functionality of ADAM to the 'lowest common denominator'.

The following components were identified as being essential for a useful data reduction system:

HDS
Parameter system
Graphics
Error system
Command language
Magnetic tape handling (MAG) (? workstations typically don't have tapes)
File I/O (FIO)

Comments:

- HDS has been implemented on the Convex (which does have file mapping) but is not actually used in the Convex Figaro.
- GKS-UK already runs on several flavours of Unix and there is an X Windows device handler running on the SUN.

- ICL is written in VAX Pascal and will probably have to be converted to C.
- The interfaces where parts of the system written in C are called by parts written in Fortran are going to be a major problem, because the relationship between the two languages is different on different computers.

There was some discussion on whether ADA would be a better language then C to translate software into as the language itself supports multi-process architectures. However, it was suspected that an ADA system has to be written as a single program with all its applications known when the system is compiled; additional applications cannot be activated at run time. Nigel Houghton agreed to investigate the properties of ADA in more detail.

7.5 Compute Servers

Session Friday 7th July – session 1

Chairman William Lupton

Programme Compute servers, transputers, VME boards. How are people using them and how do they fit into ADAM?

7.5.1 External Memory Systems

WFL began by giving a brief description of the external memory which is being procured by the AAO for use at the AAT. It is a computer system based on the VME bus and will provide the following services:

- It will read data from detectors into its local memory, potentially performing online processing.
- It will provide online display facilities.
- It will transfer data to the host machine (the VAX on which the ADAM system is running).
- It will provide simple array operations such as de-biassing and flat-fielding.

The RGO have a similar system for use at the WHT. Its hardware is similar (the AAO followed many of the RGO's hardware decisions) but its software is completely different from the AAT one. The AAT one is programmed in C and makes use of a real-time kernel called pSOS, whereas the WHT one is programmed in FORTH.

There was some discussion as to why there had been so much duplication of effort on the software side. The simple answer is that the RGO won't touch C for its microprocessor software, whereas the AAO won't touch FORTH.

Discussion then moved to the question of what processing should be performed in the external memory. If there are some sufficiently well-defined processing steps then might it not be possible to regard the output from those processing steps as being the raw data? The answer appears to be (as it always is when this discussion takes place) that whilst it may be desirable to export the processed data, the truly raw data must always be exported as well.

Turning to possible relevance of these external memory systems to Starlink, it was noted that one might be able to regard them as providing an interface at which one could plug in some of the

more computationally intensive pieces of Starlink software. However it was generally accepted that this was not really practicable since one would first have to make that software available in the environment of the external memory and in any case such systems are only expected to be available at the observatories. In conclusion, these external memory systems are not of general interest to Starlink.

7.5.2 Transputers

BDK had circulated a report on the use of transputers at ROE (see Appendix E).

Turning to possible relevance of transputer systems to Starlink, it was initially felt that they would only be worthwhile if the applications running in them exploited parallelism. However it was pointed out that this was not the case if a user could gain access to his own personal transputer – a single T800 has roughly the floating point performance and twice the double precision performance of a MicroVAX 3500. It thus appears that there could be genuine parallel and non-parallel Starlink signal and image processing applications where substantially more than MicroVAX 3500 power is needed.

For Starlink use one would take a similar approach to that of ROE with the SCUBA software development and would program in Fortran for reasons of supportability. Transputers could provide a much more cost-effective way of providing enhanced performance for specialised applications than does the purchase of VAXstations.

Part III

Submissions

Appendix A

ADAM Support Group Objectives

Submitted by: Patrick Wallace, RAL, 13th June 1989.

A.1 Introduction

At the end of 1988, Starlink made a successful bid for the creation of an ADAM Support Group (ASG). The Group will be composed of five people; two of the positions are to be drawn from existing resources and three, available from April 1990, are new, including the ASG Head. The Group will concentrate mainly on the system aspects of ADAM, plus various kernel and template applications. While supporting the real-time facilities within ADAM (including performance aspects), the ASG will leave online applications software *per se* to the observatories.

The ASG was initially funded for just three years, after which progress is to be assessed and the future of the Group reconsidered. The success of the Group will be judged in relation to a list of objectives which formed part of the original bid. This list forms the rest of this appendix.

A.2 Goals for the ADAM Support Group

General – A test of the effectiveness of the ADAM Support Group will be if it has gained the confidence of the community, so that groups developing major new systems have decided to work within ADAM. This goal will be achieved as a result of consulting the community, announcing plans in advance, and meeting promised timescales.

International use of ADAM will also be an indicator of the Group's success.

Development

- Existing code:
  - All code will be reviewed and brought into line with Starlink standards.
  - The efficiency of the entire system will be examined. Improvements, possibly substantial, may be possible in the loading of packages, the invocation of tasks, and in accessing data objects. The current level of real-time performance will be at least maintained and, if possible, enhanced.
  - The existing HELP system will be replaced by a better one.

  - The current rather unfriendly and inconsistent error reporting will be improved and standardised.

- Portability:
  - The portability state of ADAM will be studied and the objectives and strategies for creating a portable ADAM will be defined. Assuming, as seems likely, that the plans offer overall savings to the community compared with continuing to support only VAX/VMS, the Group will, with the help of university contractors, re-implement or adapt ADAM to portable form. This will almost certainly involve a port to at least one Unix machine. The initial objective will be to introduce a high level of portability in those ADAM facilities required for data analysis, but extension to real-time aspects will also be studied if this promises to be useful to the observatories.

- New code:
  - An implementation of the Image Display Interface package, IDI, will be developed and integrated into ADAM.
  - There will be a wealth of utility libraries to simplify the job of the application programmer. Of major importance will be facilities to access and manipulate standard data structures.
  - There will be improved and standardised methods of communicating with embedded instrumentation and processing elements.
  - Certain specialised techniques have grown up at the various observatories; these will be incorporated in ADAM proper. Unified techniques and standards for real time will be promoted.

Support – The various components of ADAM will be drawn together and properly integrated. Support arrangements will be defined. Interdependencies will be documented to allow new versions to be built easily. The line between ADAM itself and the real-time software produced at the observatories will be drawn.

Procedures for bug reporting, queries, and suggestions will be laid down. Arrangements will be made for software distribution through Starlink and for the distribution of urgent fixes, both to Starlink sites and to the overseas observatories.

Arrangements will be put in place for feedback from the community. The Group will operate one or more ADAM-related SIGs, and will liaise with the other Starlink SIGs. Good contact with real-time implementors and users will be maintained.

Documentation & Education – The Group will create integrated and friendly documentation, within the Starlink system but clearly packaged. There will be three main items, as follows:

- A manual for the user of ADAM applications. This will also contain information for system managers.
- A manual for user/programmers.
- Descriptions of the system architecture, documents describing how to port ADAM to other machines, and instructions for writing the IDI and GKS drivers required for new graphics devices.

There will also be a programme of education for users and user/programmers, including road shows, courses and tutorials. Teaching materials will be produced to allow site managers and contract programmers to act as instructors.

Applications – Several major applications packages will be running under ADAM and will be in wide use throughout Starlink. These will include at least the Figaro and KAPPA systems. The packages will share data formats and graphics facilities in an effective way.

Management – All staff should have been recruited and in post within 6 months of the creation of the Group.

The Group will actively market ADAM, through presentations, handouts, management briefings, and liaison with the international community.

Appendix B

Submissions from Dennis Kelly

Submitted by: Dennis Kelly, ROE, 23rd June 1989.

B.1 Introduction to Thursday, Session 1

We have three items to discuss.

- Progress on ADAM V2
- SCAR
- Figaro

In late 1987 two significant events occurred. Firstly, rumours came from AAO that Figaro had been demonstrated running as an ADAM monolith. Secondly, I demonstrated SCAR running as an ADAM monolith.

By June 1989 neither of these systems had been released in ADAM versions. I think we ought to investigate why this delay has occurred to see what we can learn from it.

AGENDA

1. ADAM V2
   Progress towards ADAM V2 Dennis Kelly
   Discussion

2. SCAR
   Progress on SCAR Dennis Kelly
   Questions from the database SIG Dennis Kelly
   'Some musings on ADAM and ADC' (Clive Davenhall)
   Discussion

3. Figaro
   Progress on Figaro Keith Shortridge
   Discussion

B.2 Progress Towards ADAM V2

ADAM Version 2 uses Jeremy Bailey's MSP package instead of VMS mailboxes for low-level inter-task communication. It also supports sending messages across DECnet and hooks have been designed-in to allow extended control of one task by another in due course.

The initial ADAM V2 is implemented to be compatible with V1. It essentially consists of a re-implementation of the inside of the MESSYS library. As a result, it is possible to switch between V1 and V2 by changing the logical name ADAMSHARE. It is not necessary to relink tasks. The V2 code was completed and subjected to 'verification' by code reading in mid-1988, and has been available for evaluation since then. Code reading of MSP was carried out in June 1989.

There is now a strong demand for V2 to be available as a released system within the next couple of months. This is primarily because of its networking ability.

Future developments in the networking area are likely to include:

- Access to transputer systems.
- Copying of noticeboard (NBS) data.

B.3 Progress on SCAR

SCAR is a set of relational database applications implemented on top of a library called ADC. The SCAR/ADC system was implemented by Jon Fairclough. It was designed to be compatible with both the INTERIM environment and the SSE. It did this by providing its own parameter system routines which called the underlying environments. In late 1987 I provided the modifications formally necessary to make SCAR run under ADAM. This was demonstrated at the Database SIG in December 1987.

Two outstanding areas of work remained:

1. Designing IFLs to make the prompting user-friendly.
2. The applications showed unexpected behaviour on the second and subsequent invocations after loading.

I then had to stop work on this. Sandy Leggett, then the database programmer, was able to continue the work after a delay of three months or so.

Problem (2) was found to be due to uninitialised Fortran variables in the SCAR code. In other words, the SCAR programs were OK running to completion as INTERIM applications, but were not satisfactory loaded once and executed many times as ADAM tasks are.

The problems were mostly solved by the time that Sandy left in August 1988, although a new variant of (2) attributable to NCAR had surfaced. However, the software was not releasable – it required further tidying and documentation. The software was shipped to the IPMAF group at RAL, no further effort being available at ROE.

David Giaretta managed to get the software up and running and became sufficiently familiar with it to make some minor changes, but did not have time to do more. In January 1989 Jon Fairclough re-started work on it. Naturally, his main interest was in adding new features as

required by the project he was working on. He completed this work and brought the system close to the point of being releasable, but left in mid-June 1989. The IPMAF group are now trying to complete the work of getting the software released.

So, why has it taken 18 months to go from a demonstration to a releasable system?

Answer – probably 6 months are attributable to genuine software problems. The rest boils down to lack of continuity in manpower.

B.4 Questions from the Database SIG

The DATABASE SIG has identified three issues which the Workshop might like to consider.

Organisation of releases A large part of SCAR consists of items such as the ADC library, which provides a subroutine interface for database applications. The nature of this code is akin to ADAM 'system' code. It might make sense, therefore, to accept SCAR/ADC as a standard ADAM component and organise releases and directory structures accordingly.

Relationship between SCAR and ICL To what extent should we integrate database operations with the command language?

Controlling screens from within applications Some SCAR applications which can produce attractive screen-formatted output. The current ADAM user interfaces don't have a facility for 'handing' the screen over to an application in a neat way. This is an ability that may be necessary in the future.

B.5 Transputers at ROE

ROE have a pilot project using transputers as part of a feasibility study for a replacement for COSMOS. This project has an INMOS system containing about 20 transputers interfaced to an IBM-PC compatible (VAXMATE). The system is programmed in Occam, and is designed to handle the real-time processing of COSMOS data.

The first ADAM-related transputer work at ROE is being carried out as part of the SCUBA project. SCUBA is a submillimetre array instrument being built for the JCMT. The delivery target is 2.5 years away. The real-time computer for SCUBA will be a transputer system which will carry out signal processing and image processing. It will be interfaced to a VAX running ADAM. Commands will have to be sent from the ADAM system and data received back.

We have a Caplin Cybernetics Corporation transputer system interfaced to a MicroVAX II which we are using for initial development. The software will be implemented in parallel Fortran thereby easing subsequent software maintenance. Communication between the VAX and the transputers uses the ADAM V2 networking protocols, but currently requires a D-task in the VAX to do the communications. Eventually this ability will be built into the ADAMNET process.

Appendix C

Some musings on ADAM and ADC

Submitted by: A C Davenhall, ROE, 22nd June 1989.

Abstract

The relation between ADAM and ADC is considered and the following conclusions are drawn:

1. ADC forms a natural part of the ADAM environment.
2. The ADC subroutine libraries should be fully integrated with ADAM and maintained by the ADAM Support Group.
3. There are no significant advantages in re-implementing ADC to access catalogues held inside HDS structures rather than as VMS files.
4. SCAR applications should be free to use the full ADAM parameter system.

C.1 Introduction

With the preparation and imminent release of a new version of SCAR (V5.1) that runs under the ADAM environment it is timely to consider the relation between SCAR/ADC and ADAM. This relation is of genuine practical concern, rather than mere academic interest, because packages are being planned which make full use of the ADAM data reduction environment, including use of HDS, and which also manipulate ADC catalogues. Such applications are planned, for example, at Birmingham as part of the software for the ROSAT satellite and at ROE as part of the software for SCUBA, an array instrument for the JCMT (see *e.g.* ref. [1]).

C.2 SCAR and ADC

The SCAR package was produced four or five years ago by IPMAF as a general package for handling relational data, but with additional astronomical functions, and in particular functions for handling the IRAS catalogues. The basic SCAR system can be seen as a general database

system to which have been added a quite extensive set of astronomical functions, including for example the ability to manipulate astronomical coordinate systems. Thus in the area of databases SCAR fulfills a rôle similar to that of KAPPA in the area of image processing. It is envisaged that more specialised applications, for processing particular types of data or for work in a particular branch of astronomy will 'sit on top of' or 'co-exist with' the basic SCAR system, rather than duplicating its functionality. The applications planned for ROSAT fall into this category.

Within the SCAR package the ADC routines provide facilities for accessing and creating catalogues and for performing database operations on relational datasets. That is, SCAR is a set of applications whereas ADC is a subroutine library providing basic catalogue handling functions (at least that is the ideal division of functionality – in practice the separation is not quite so clear cut). Thus the ADC routines fall firmly on the 'system' side of the 'applications – system' divide. To continue the analogy between SCAR and KAPPA, the role of ADC in SCAR is similar to that of HDS in KAPPA. The previous ADAM workshop endorsed SCAR as the standard relational database tool for the ADAM environment [2].

The original version of SCAR was designed to be linked with either the original SSE (effectively the precursor to ADAM), the interim environment or as 'stand-alone' Fortran programs. However, it was developed and released by linking with the interim environment. This degree of 'environment independence' was achieved by designing a small set of subroutines which carried out all the 'environment' functions required by SCAR. When a SCAR program wanted an 'environment' service it would call one of these subroutines rather than calling the environment directly. These subroutines were then implemented using calls to the SSE, calls to the interim environment and as 'stand-alone' Fortran respectively. This approach was only practical because the original SCAR programs made rather limited demands on the environment, effectively limited to obtaining and outputting parameters and outputting messages.

C.3 Future Applications

As mentioned above applications are already planned as part of the ROSAT and SCUBA projects which will use both HDS and ADC. ADC is the natural way to handle lists of tabular data, for example a table of stellar positions and magnitudes produced by running a photometry package on a CCD frame is best represented as an ADC catalogue.

Within these applications the ADC routines function like any other library in the ADAM environment and are called when their functionality is required, alongside calls to HDS, to the parameter system *etc.* Thus there is nothing special about the ADC library and it should be fully integrated with the other ADAM libraries and it is appropriate that it should be maintained as part of ADAM by the support group.

C.4 Maintenance and Development

The standards of coding, the underlying philosophy, *etc.* of the ADC routines are very similar to those of ADAM. It is largely because of this similarity that integration into ADAM is practical and it has the further consequence that programmers can switch between ADC and other ADAM work without much retraining or re-orientation.

The following are suggested as possible items of work that are required on ADC:

1. There are extensive facilities in ADC, but better documentation is required if they are to be used to the full. What is required is not so much improved documentation on individual facilities and routines, but an improved way of indexing and arranging what is available in order to expedite locating the routines and facilities which might be useful to solve a particular task.

2. The current SCAR/ADC system has its own copies of some of the standard ADAM libraries, such as CHR, MIO and FIO. The package should be changed to link with the standard versions of these libraries. However, it is important that the bug fixes and enhancements made to the SCAR versions of the libraries should be propagated back into the standard libraries. Only having one version of the libraries is of considerable importance in order to simplify their future maintenance.

3. The standard ADAM mechanism for aborting applications by supplying a null parameter value is absent from the current version of SCAR running under ADAM. The SCAR applications should be made to behave like all other ADAM applications in this respect.

The first two items of work fairly obviously fall within the remit of an ADAM support group supporting ADC. In the case of the third item it would be necessary to determine whether the problem was with the SCAR applications or the ADC routines. There are a number of other similar, relatively minor, problems that need to be fixed.

C.5 Parameter System

As explained in section 2, the ADC library effectively contains its own parameter system, implemented as a 'wrap-around' of the ADAM parameter system (and previously as a 'wrap-around' of the interim environment parameter system). The current SCAR applications call this 'ADC parameter system' exclusively, *i.e.* they contain no direct calls to the ADAM parameter system. This approach was effective in reducing the amount of work in transferring from the interim to the ADAM environment, but the 'ADC parameter system' contains only a few routines and does not have the full functionality of the ADAM parameter system. For example dynamic defaults for parameter values are not available. Nor are tokens supported, so if messages are to be reported containing the values of variables they have to be laboriously constructed using the CHR routines directly. This approach leads to unnecessarily long-winded code (albeit code that is easy to understand).

Now that SCAR runs under the ADAM environment the question arises as to whether new applications should obtain parameters using the 'ADC parameter system' or whether they should call the standard ADAM parameter system directly. Given that applications that use ADC are now just ADAM applications that happen to call the ADC library there is no reason why they should not call the ADAM parameter system directly, like any other ADAM application. By adopting this approach the full power and flexibility of the ADAM parameter system would be available to them.

C.6 ADC Catalogues Inside HDS Structures

It would technically be possible to re-implement ADC to access catalogues held inside a HDS structure rather than as VMS files (or tape files). Such an approach would not lead to an increase in speed because access to the catalogues is not normally I/O bound (see *e.g.* ref. [3]).

Formally, in the present implementation, it is a property of ADC catalogues that no restrictions are placed on their length; this property would be lost if they were to be held inside a HDS structure, though the difference is unlikely to be important in practice. A dataset consisting of a mixture of catalogues and other data items could all be held in a single container file rather than two (or more) catalogue files and a container file for the other items; this arrangement has both advantages and disadvantages.

Holding ADC catalogues in this way would make it more difficult to access them using programs that do not run under the ADAM environment. This consideration is important for a number of reasons:

- Many large catalogues are not written using ADC. They are obtained from some external source already written in some format and are 'converted' into ADC catalogues by creating an ADC description file for them.
- Some special purpose applications that only work on a specific type of catalogue 'cheat' by ignoring the ADC interface and reading the catalogue with Fortran I/O in order to obtain better performance.
- The present ADC implementation allows catalogues to be read directly into some external software packages, such as CLUSTAN.

All these facilities would be lost if ADC catalogues were held as HDS structures.

Thus there are no compelling advantages to holding ADC catalogues in a HDS structure, but there are a number of disadvantages. In addition it is anticipated that this feature would be tricky and time consuming to implement (see *e.g.* ref. [4]).

C.7 Conclusions

The ADC facility is a natural part of the ADAM environment and is best viewed as just another facility, like HDS, or the parameter system or any of the other libraries. From this approach it follows that:

1. ADC is an appropriate item for the ADAM support group to maintain.
2. Future ADAM applications that use ADC should be free to use the full ADAM parameter system.

There are no obvious advantages, but several disadvantages, to re-implementing ADC to access catalogues held inside HDS structures. The present version of SCAR contains copies of several standard ADAM libraries; these should be discarded in favour of using the standard ones, but the bug fixes and enhancements they contain should be propagated into the standard libraries.

C.8 Bibliography

[1] A C Davenhall, 1989, *The SCUBA data structures: a first guess*, SCU/4.0/ACD/689, ROE internal note.

[2] J H Fairclough (ed.), 1988, *The proceedings of the 1987 ADAM workshop*, Chap. 1, section 1.3.

[3] A C Davenhall, 1986, *I/O Timing tests on the catalogue handling systems*, ROE internal note.

[4] A C Davenhall, 1989, *Minutes of the meeting of the Starlink database SIG held on 2nd June 1989*, Section 8.2.

Appendix D

Suggested Improvements to the Parameter system

Submitted by: A.J.Chipperfield, RAL, 19th June 1989.

D.1 Introduction

There are a number of anomalies in the way parameters are obtained, leading to confusion amongst programmers and users alike.

The two main areas of concern are:

- The way in which parameter state affects the value obtained.
- The way in which the 'current' value is defined.

D.2 The Affect of Parameter State

D.2.1 Introduction

Parameter states affect the way in which parameter values are obtained. This process is described in AED/15 although the description is out of date in that it does not describe the pseudo GROUND states which are:

- SUBPAR__RESET – Follow vpath - ignore 'current' on ppath.
- SUBPAR__ACCPR – Use the prompt value.
- SUBPAR__RESACC – RESET and ACCPR.
- SUBPAR__FPROMPT – Force a prompt.
- SUBPAR__RESPROM – RESET and FPROMPT.

Furthermore, the descriptions lack some detail.

D.2.2 GROUND and Pseudo GROUND States

SUBPAR__GROUND is the initial state of a parameter and the state it is left in after the parameter system has been de-activated at the end of an Atask.

In this state, if a value for the parameter has not been provided on the command line (or by means of previous SET context), the vpath is searched to obtain a value.

A parameter may be switched from GROUND to a PSEUDO GROUND state by means of the RESET, ACCEPT and PROMPT keywords on the command line.

In the pseudo GROUND states, the vpath search is modified as indicated above.

D.2.3 CANCELLED State (SUBPAR__CANCEL)

Following PAR_CANCL the parameter is in the CANCELLED state. Currently, a subsequent attempt to obtain another value for the parameter will follow the vpath but ignore 'dynamic', 'default' and 'global' specifiers. Specifiers 'noprompt', 'prompt' and 'current' operate the same as for the GROUND state.

D.2.4 Suggested Improvements

Action in CANCELLED state.

It is suggested that the vpath should be totally ignored and a prompt forced. This would affect certain vpath specifiers as follows:

1. noprompt – prompt (currently returns status PAR__NULL).
2. current – prompt (currently takes the 'current' value).
3. prompt – prompt (no change).
4. default – prompt (no change).
5. dynamic – prompt (no change).
6. global – prompt (no change).
7. internal – take default (no change).

Effect of RESET state.

It is suggested that RESET state should cause a 'current' specifier to be ignored on either the vpath or ppath. At the moment, it only affects the ppath.

Setting the parameter state

A requirement has been found for the application to be able to:

1. Follow the vpath again rather than prompting after cancelling the parameter.

2. Attempt to get another value for the parameter ignoring the 'current' value (if it is not suitable for example).

To achieve this, and possibly other requirements which haven't yet shown up, it is proposed to provide subroutines PAR_CANST and SUBPAR_CANST, equivalent to the _CANCL routines but allowing options on the state into which the parameter is to be set.

```
*+  PAR_CANST - Cancel a parameter into a specified state.

*    Description :
*     Any existing association between the parameter and a data
*     system object is cancelled, and the container file for it closed.
*     The parameter is then placed into the specified state.
*
*     This enables the application to cancel a parameter but follow
*     the vpath again rather than prompting to obtain a new value, or
*     to cause a 'current' value to be ignored.
*     If STATE is SUBPAR__CANCEL, the effect is the same as calling
*     PAR_CANCL.
*
*     Parameter states are described in AED/15 and symbolic constants
*     defined in the Fortran INCLUDE file 'SUBPAR_PAR'.
*
*     The following states may be specified:
*           SUBPAR__GROUND   Follow vpath on next PAR_GET.
*           SUBPAR__CANCEL   Prompt on next PAR_GET - As PAR_CANCL.
*           SUBPAR__NULL     Give null response on next PAR_GET
*           SUBPAR__RESET    Follow vpath - Ignore 'current' on ppath.
*           SUBPAR__ACCPR    Use the prompt value.
*           SUBPAR__RESACC   RESET and ACCPR.
*           SUBPAR__FPROMPT  Force a prompt as PAR_CANCL would.
*           SUBPAR__RESPROM  RESET and FPROMPT.
*
*     If an illegal state is specified, the parameter will be cancelled
*     and STATUS set to SUBPAR__INVST.
*    Invocation :
*     CALL PAR_CANST ( PARAM, STATE, STATUS )
*    Parameters :
*     PARAM=CHARACTER*(*) (given)
*           parameter name
*     STATE=INTEGER (given)
*           required state of parameter
*     STATUS=INTEGER
*           Variable holding the status value. The routine is executed
*           regardless of the import value of STATUS.
*           If the import value is not SAI__OK, then it is left
*           unchanged, even if the routine fails to complete.
*           If the import value is SAI__OK on entry and the routine
*           fails to complete, STATUS will be set to an appropriate
*           error number.
```

PAR_CANST should be used with care as it could result in infinite loops if the vpath is used repeatedly and always gives the same erroneous value.

Also, it is violating the principle of separating the application from the outside world and making the application dependent upon the way the environment is implemented.

D.3 Current Values

D.3.1 The Problem

The 'current' value of a non-internal parameter is actually the value stored in the task's private parameter storage (ADAM_USER:task.SDF). At the moment, if a primitive value is given for a parameter, it is stored immediately and becomes the current value for any subsequent parameter GET. However, if a name is given as the value, it is only entered into the task's private storage at the end of the task and then only if the parameter is in the ACTIVE state and the task ended with status SAI__OK.

If:

1. X is a parameter of task TEST (which loops, obtaining a value for X, printing it then cancelling X).
2. X has vpath 'prompt' and ppath 'current'.
3. The value of X used on the last invocation of TEST was 3.3.
4. NUMBER is a primitive HDS object containing 5.5.

the following effect would be observed:

```
ICL> define test test
ICL> test
X/3.3/> 4.4
test prints 4.4
X/4.4/> number
test prints 5.5
X/4.4/>

   etc.
```

i.e. when the object name is given, it is not remembered, but when a primitive value is given, that is remembered.

Furthermore, if a primitive value is given and the task decides that the value is outside the range it can deal with, it would offer that unsuitable value as the next prompt value.

D.3.2 Suggested Improvement

Names should also be entered into the parameter storage immediately they are obtained.

This has the disadvantage that unusable names (*e.g.* file not found) will be entered in as 'current' values and thus could be offered as prompt values. The advantage would be that the same rules apply in the case of names or primitive values.

Where the parameter system itself decides to ask for another value, it should be possible to use the SUBPAR_CANST routine, proposed above, with state SUBPAR__RESET to prevent a bad current value being offered. The application could do this too, if it decided that the 'current' value should be ignored. However, this would not prevent a bad value being saved for the next invocation of the task if no subsequent good value is obtained. The only way to do that would be to 'un-define' the parameter storage in some way. In either case, the previous good 'current' value would already have been lost.

Appendix E

Time to Raise Cain? – A critique of the VAX ADAM system

Submitted by: Lewis Jones, RGO, 30th June 1989.

E.1 Introduction

The VAX ADAM system presently used by ROE, AAO, Starlink and others has been used to build the instrumentation systems for the William Herschel Telescope (WHT). Use of 'intelligent' microprocessor-based hardware within the WHT observing system, together with a fundamental requirement for increased efficiency in the observing process, has resulted in a new approach requiring sophisticated software in order to implement batch-oriented control. The result is that the WHT instrumentation system forms one of the largest and most complex implementations within the ADAM environment to date.

During the design and implementation stages of the WHT instrumentation software, the ADAM environment has been found to be a less than satisfactory framework within which to construct a high-quality observing system to the original specification. This document is a distillation of specific problems which have been encountered by members of the La Palma Software Group, together with other general observations of shortcomings in the current implementation of the VAX ADAM environment.

E.2 Background

ADAM was initially developed at RGO to provide an instrumentation control environment for the Isaac Newton and Jacobus Kapteyn telescopes. When ROE and MRAO decided to adopt the ADAM architecture for data acquisition at the UK Infra-Red and James Clerk Maxwell Telescopes, Perkin-Elmer ADAM was ported to VAX/VMS. The original concepts and nomenclature were retained, but the code was rewritten incorporating material from the Starlink Software Environment. VAX ADAM was then adopted by the Anglo-Australian Observatory (AAO) for instrument control and by Starlink as a data reduction environment. At the time RGO re-adopted ADAM for control of the WHT (Autumn 1985), instrument development was well under way and a specification of the interface between the VAX Instrumentation System Computer was available. However, the original specification of ADAM as a control (rather than reduction) environment has changed very little between inception on Perkin-Elmer processors

and maturity under VAX/VMS, and doesn't encompass current requirements *vis à vis* network access to instruments, information display and user control. Rather, efforts in these directions are tending towards 'add-on', specific, solutions addressing individual needs.

Appended below is a brief description of the environment within which common-user observations is obliged take place on the WHT. For simplicity in this section, the term 'instruments' is used to describe any processor attached to the Utility Network, including the telescope, detectors and image display processors (like the DMS).

E.2.1 The WHT Environment

The WHT's instruments are interconnected by an Ethernet 'spine' and are normally based on the 6809 processor (known locally as 4MS) or 68008/68020 VME processors. For example, the Detector Memory System (a generic detector image storage and manipulation subsystem replacing the old External Memory) is based on a 68020 processor. Commands are sent to an individual processor on the network using an RGO protocol in accordance with the IEEE 802.2 standard. The physical and data-link layers of the ISO 7 layer Reference Model for Open Systems Interconnection are implemented in hardware using Sension Network Interface Units. The transport and application layers are handled by software running on the attached processors. The protocol involves an attached processor announcing its presence by a broadcast message and then being able to communicate on a one-to-one basis with any other processor on the network. Note that only commands and status are interchanged in this fashion, image data is transferred by a private, parallel link.

On the VAX 3600 System Computer, any process wishing to communicate with an instrument sends a message (using the ADAM MSP system) to the Utility Network (UTILNET) process. This transmits it onwards, guaranteeing its receipt by the destination processor. In general a number of replies will be expected in return. These are: an ACCEPTED message (indicating that the request has been started), and a COMPLETED message (indicating completion and carrying any requested data with it) with interspersed PROGRESS messages containing updated status values during a long action. When no action is outstanding, a processor is at liberty to send MONITOR messages (if they have been enabled) to indicate a state which has changed (*e.g.* mechanism position drifts or telemetry value goes out of range). These responses are received by UTILNET and forwarded to the appropriate process.

It should be noted that this means of communication with an attached processor is somewhat different to the protocol used in previous RGO ADAM systems, where the equivalent System Computer communicates directly with the processor using CAMAC in a synchronous fashion.

E.2.2 The Control File System

Traditional ADAM systems have relied on more or less direct control from the user. User commands of the form `SEND IPCS OBEY EXPOSE` are certainly possible, if not actually encouraged. Even where a layer of processes is interposed between the user and the D-task level, the interaction tends to mimic a user typing at a keyboard. For the WHT, the top level isn't an astronomer; instead, a process called the Control File Executor (CFX) reads a description of the observations required and converts this into a stream of essentially parallel commands to multiple instruments. While being driven by state information returned by the lower echelons of the system, the CFX is able to draw on a detailed description of how the system functions (the 'Mechanism Database') in order to determine the most efficient sequence of commands to effect each set of state transitions in an observing sequence.

E.2.3 The WHT ADAM System

The WHT ADAM system began about three years ago with a decision by the La Palma Software Group to adopt VAX ADAM as the production environment for data acquisition and instrument control. Shortly afterwards, a number of environment-independent analyses were written using Yourdon design techniques which defined the elements of the system as it appeared at that time. After a number of staff left the software group between July 1986 and July 1987, effort was (on average) concentrated on the WHT telescope control system with a (somewhat smaller) group working on instrumentation. A re-appraisal of the requirements for WHT instrumentation control led to the careful re-analysis in late 1987 of the initial user requirements. Following on from this, functional specifications were produced for all instruments. Newly recruited members of the La Palma Software Group spent some months learning about the ADAM environment and pronounced their dissatisfaction at the difficulty of developing applications within it. Indeed there was much questioning of the general suitability of ADAM for the task which lay ahead. Given the short timescale between then and completing a running system[1] by Autumn 1988, I decided that we had to make the best use of what was available. In retrospect it is almost certainly true to say that a better system could have been produced with less effort by using VMS together with certain carefully chosen parts of the ADAM system, (*e.g.* the noticeboard system).

The remainder of this document enumerates the difficulties which have beset the La Palma Software Group in their use of ADAM during the past 12 months or so. Many of the problems surfaced during the implementation phase and would only have been spotted ahead of time by experienced ADAM programmers; the nature of these problems indicate that ADAM in its present form is difficult to use and is unsuited for use in a major software engineering project.

E.3 The problems with ADAM

E.3.1 General Software Support

The state of support for the kernel code of ADAM is not acceptable at the present. ROE have been obliged to withdraw support and little if any effort has been available from central Starlink resources. A better plan of support is required, using the distributed resources of prime ADAM sites where necessary. This must be less *ad hoc* than the 'let the user fix his own problems and we'll review the solution' approach if ADAM is to survive and grow properly as an acknowledged major contribution to astronomical computing. The proposals for an ADAM Support Group presented by Starlink to the Theory and Computing Panel of the APS board will help to resolve the situation so long as adequate consultation is maintained with users and implementors.

E.3.2 Support for Specific Hardware

As far as the author is aware, RGO owns the only Starlink-compatible multiprocessor system – a VAX 8300. VMS V5.0 has required changes to system software for proper handling of multiprocessors. Does this mean that the RGO VAX 8300 is unsupported if, for any reason, ADAM software doesn't work on an 8300 and the reason is attributable to the hardware? What happens with future VAXes, with Local Area VAXclusters *etc.* and how much support, if any, does Starlink guarantee?

[1] For the WHT Faint Object Spectrograph. This was chosen as the 'test-bed' to try out the infrastructure (Utility Network, Detector Memory *etc.*) in preparation for commissioning the ISIS triple spectrograph.

E.3.3 Quality Control

There are a number of real and potential areas of concern regarding quality control in the production of ADAM Systems Code. In particular, there needs to be a professional approach in such areas as upwards compatibility, code and design clarity, state of documentation and control of upgrades.

1. **Standards of Kernel Code**

   Given that end-users are obliged to perform some maintenance of Starlink systems (of which ADAM is now a part), it is essential for code which is adopted as part of ADAM to be produced to a high standard. It is certainly true that some is indeed well-produced (although this is a subjective term) but there are disparate styles.

   Nevertheless, some ADAM system code is

   - Inadequately commented or even uncommented.
   - Poorly laid out with little or no header information.
   - Generally difficult to assimilate.

   The Starlink Fortran Standards document (SGP/16) has helped to improve Fortran code, and equivalent documents for Pascal and C would certainly help in the future production of non-Fortran code. However, it is also true that SGP/16 (and probably any proposed Starlink standards for Pascal and C programming too) is more oriented towards good applications programming. System-level code is likely to be more complex and should be coded to an even higher standard. Seamless joins and common interfaces between the different parts of ADAM are vital and only derive from properly considered designs.

2. **Code Testing**

   ADAM is a highly complex suite of programs and code libraries and is therefore difficult to test exhaustively. However, basic confidence checks need to be performed every time any part of the code is changed.

   An example concerns the MAG_ALOC routine, which when called causes the calling program to crash with an access violation because the address of the status parameter given to HDS_OPEN is corrupt (value 6). Simple checking using, for example, DEC/Test Manager should cure obvious problems such as routines which don't work at all.

   In addition, a number of significant problems have been found in the version of MSP as originally released. These were serious enough to mean that the La Palma Software Group had to spend time finding solutions which would enable the WHT systems to function properly. The fixes devised by the software group have now been passed on to Jeremy Bailey to be incorporated in the released version.

3. **Documentation**

   The ADAM documentation kit is rather like the curate's egg - good in parts; however, its complexity understandably makes it difficult to document fully with limited resources. Comparison against the many well-written volumes of the VAX/VMS documentation kit is of course unfair, but something of this calibre would be a worthy goal to achieve.

   To return in particular to the MAG_ routines, ADAM documentation can be very misleading or even uninformative, simply because it has been written by experts. In one particular case, the operation of MAG_ALOC was not fully understood by a programmer

from the documentation. A prior call to GET_PAR0C to get the name of the tape deck from the IFL resolved this parameter unnecessarily. MAG_ALOC then looped issuing an error message '! No error to report (faulty application)'. Fine and, in retrospect, true, but how is the ADAM novice supposed to cope with uninformative messages such as this, especially if they are not documented?

4. **Upwards compatibility**

   It is important that ADAM sites should not have unnecessary work thrust upon them when a new version of the ADAM system is released. Unfortunately, this is not always the case; a particular example is the new version of the Noticeboard System (NBS V2.2) which has a changed interface to certain user-level routines. In this case, since the changes are modifications to the number and type of parameters, upwards compatibility could have been maintained by providing new routines with the full capability, and retaining the old routines (possibly based around the new ones) with limited, but compatible capability.

5. **Ease of Use**

   The ease of production of applications programs by non-systems programmers (and ideally by people who have little, if any, prior knowledge of ADAM) needs to be addressed. At present it is rather too difficult to produce ADAM applications (which may explain the general lack of new ADAM applications and the espousal of existing Figaro applications instead).

6. **Problem reporting**

   It would help both programmers and users if a central repository of Software Performance Reports (SPRs) was to be kept and a formal method instituted of noting problems and their fixes. People should then be able to interrogate this database and to submit new SPRs. Requests for modifications and new facilities would need collation and review by an arbiter before being allocated effort.

7. **New Releases**

   It would help those installing new versions of ADAM if a list of modifications made to each sub-system were available in a set of release notes.

8. **Efficiency**

   Efficiency of implementation and clarity of code do not have to be traded against each other. Therefore, given the use of ADAM within on-line systems, efficiency should be addressed and improved wherever possible. A neatly programmed system which is slow may be acceptable; one which is neither neat (taken as a whole) nor efficient certainly is not.

E.3.4 ADAM Processes

1. **Process types**

   ADAM presently 'supports' a proliferation of different flavours of process (*e.g.* B, C, CD, D, A, M, U) all of which are actually based on the same D-task kernel. It can be difficult, particularly in a control environment, to decide between task types (*e.g.* between D, CD and U). The features which are selected by the choice of task type should be individually selectable from the interface file. The proliferation of ADAM process types should be addressed at the ADAM architecture level and made unnecessary.

2. **Resource requirements**

   The requirements of ADAM processes for scarce system resources (memory, CPU cycles *etc.*) should be examined carefully and improved where possible. For example, D-task memory requirements are rather large compared with the functionality available.

3. **System integration**

   A current ADAM system is an amorphous collection of processes. It would be better (and probably would result in improved system reliability) if ADAM processes were integrated more closely. Each ADAM process should have a set of 'public' process information items held in a commonly accessible place. See also the section on AST handling. Packages would also benefit from added integration and functionality – *e.g.* specification of noticeboards in textual form integrated with the noticeboard system.

4. **Initialisation**

   The initialisation of ADAM processes and that of an ADAM system *in toto* needs attention. DEVINIT is no longer satisfactory by itself; D-tasks at least would benefit from a known, standard, initialisation action which was performed automatically after DEVINIT (selectable from the IFL in preference) and which could be initiated by OBEY or AST action to invoke a 'warm-start' at a later time. This type of feature is used to good effect in the WHT telescope control system.

5. **Process Termination**

   All ADAM processes should declare an exit handler (in supervisor mode) so that a catastrophic process failure (STOP or EXIT) is caught and can be reported to the parent, and, if necessary, emergency action taken to safeguard the instrument or detector. *n.b.* User mode exit handlers are not executed if a process is STOPped, however exit handlers declared in Supervisor and Executive modes are executed prior to image rundown.

6. **System shutdown**

   A means (preferably a standard action and AST routine) should be provided to close down (a) a single D-task and (b) an entire ADAM system **in safety**. The result after doing so should be a well-defined system state which does not compromise the instrument or detector systems.

7. **AST handling**

   (a) **General Support**

   ADAM support for AST handling is minimal and needs revision urgently. It must be possible to cope with user-defined ASTs as well as system-oriented traps.

   (b) **Library Support**

   Kernel and applications libraries should not disable AST handling and re-enable it without checking the original state on entry (MSP, for example enables and disables ASTs without re-instating the *status quo*).

   (c) **Public ASTs**

   A set of public ASTs for performing well-defined actions such as initialisation and shut-down should be provided.

   (d) **The DTASK_ASTSIGNAL routine**

   A typical D-task action to move a mechanism is a set of staged sub-actions (phases). In the first phase the D-task sends the MOVE command and then de-stages itself (by

exiting with ACT__ASTINT), waiting to be re-staged when a message arrives for it from the 4MS. The routine supplied for this is DTASK_ASTSIGNAL – it is called by that section of the D-task which deals with incoming messages, in order to re-stage the action to which the message pertains. If DTASK_ASTSIGNAL is called a second time before the first action has been re-entered, the first action is never re-staged and no error message is given. This problem is side-stepped at the moment by:

- Disabling AST delivery after calling DTASK_ASTSIGNAL until the D-task kernel code is re-entered after leaving the ACT routine. No user-mode ASTs can be delivered within this time window.
- Insisting that DTASK_ASTSIGNAL is called only within ASTs.
- Forbidding more than one call to DTASK_ASTSIGNAL per AST.

This imposes severe constraints on the design of D-tasks, and means that in some areas the system's potential cannot be achieved – for example, multiple status returns can be packaged in a single Utility Network message, but ADAM doesn't allow them to be handled in this way.

Use of ASTs within VMS allows neat, interrupt-driven programs. ADAM seems to make it very difficult to make good use of this facility.

8. **D-task actions within a D-task**

It should be possible to queue actions belonging to a D-task from within the same D-task. This should not beg problems of subprogram re-entrancy in the case of Fortran, so long as the actions are queued until the end of the currently executing action. The concept of an action queue would be invaluable here – one could add to the bottom or the top of the queue depending on the urgency of the action. VMS and the WHT telescope control system use queueing widely to good effect. Interlocking of commands is implicit here in that starting one action may imply locking itself (where queueing is not appropriate) or other actions.

9. **Debugging D-tasks**

D-tasks are notoriously inconvenient to debug using the VMS symbolic debugger in the context of a running system, because they are sub-processes of ADAMCL (or ICL) and I/O to/from the debugger conflicts with that of the parent process.

10. **D-task kernel inactivity**

The D-task kernel is inactive during the execution of an ACT routine, except between the defined stages, meaning that SETs and GETs (or for that matter CANCELs) cannot happen. If the execution of a particular stage is tardy then this could pose problems to the Control File system while it prepares for other actions on the same D-task.

11. **SEND to a hung D-task**

The current situation is that if a D-task is 'hung' within the ACT routine and an innocent process performs an ADAM_SEND to it, then it too hangs. A timeout mechanism is implemented, but the value of $\simeq$ 20 seconds is too long within the context of the WHT Control File System. It would be useful to be able to issue GETs and SETs in this state for debugging purposes if nothing else. Nevertheless, the state of the calling process shouldn't be compromised by the timeout state (*i.e.* it should be possible to obtain an immediate response if necessary)

12. **D-task status returns**

At present, the D-task protocol expects that any event which could 'unexpectedly' return status should be the subject of a special action which is outstanding at the the time of the event and which completes in response to it. Status is thus delivered to the caller who may or may not re-issue the OBEY to cope with subsequent status returns. All events are thus, by this artefact, synchronous. A less contrived and artificial way of handling this return of asynchronous status must be sought. An obvious way within VAX/VMS is to allow the D-task to queue an AST to an appropriate task (or tasks) with an associated status description block (see above).

13. **Action queueing**

For certain OBEY commands, it is almost essential to support the concept of issuing a given command for a second (or subsequent) time while the original command is still active. One example of this is issuing a second telescope slew command without an intervening CANCEL which would cause the telescope to stop first.

VMS provides excellent queueing facilities which are not used as part of the ADAM architecture.

14. **Action name checking**

Within a running, debugged system it should be possible to turn off many of the routine D-task checks. Most of these are not required except in a directly interactive system. Where ADAM SENDs are wrapped up within other tasks, ICL procedures or issued by the WHT Control File system, this checking is inclined to be wasteful, particularly if the checks are already being done elsewhere. In the specific case of action name checking, very little advantage accrues from having this done by the D-task kernel, simply because the D-task author has to recheck the action name in order to identify the action to be performed by the ACT routine. The only guarantee offered by the D-task kernel is that the ACT routine will not be called with an invalid action name, thus relieving the author from coding a catch-all clause into the action routine selection code.

The D-task kernel appears to use a linear search algorithm to check action names (and presumably to identify SET/GET variables names) which could be improved for complex D-tasks (with many actions) by using a hashing algorithm.

15. **Receipt of action completion messages**

The controlling task cannot receive D-task action completion messages asynchronously because of limitations in the ADAM_GETREPLY and ADAM_GETREPLYT routines. If a caller issues (say) 10 OBEYs to separate D-tasks then the handshake with the destination task(s) has to involve 10 calls to ADAM_GETREPLY(T). The 'T' version allows a timeout on receipt of the handshake (completion) message allowing a loop to be constructed to receive the messages while other processing is in progress. If ADAM_GETREPLY is used, then no other processing can be done and, if no message is received (say the D-task hangs), then the caller also hangs up. This system is wasteful of resources and incurs more coding effort than the use of an AST-driven mechanism.

Recent proposals by Dennis Kelly should help to address this problem.

16. **Error handling**

Error handling and logging needs to be reviewed. Error messages are normally returned as strings using ERR_INFORM. This means:

(a) An overhead in passing a string back to the caller.

(b) A fixed maximum length of 80 bytes.

(c) Discrepancies in message format between authors – the user interface is then not seamless.

(d) Insufficient contextual information for system programmers to locate the problem and fix it.

(e) Difficulties for the user in having to comprehend a stream of error messages which may or may not be related.

(f) A more or less fixed error severity level for a particular message. This is certainly better than nothing, but some errors can be more severe depending on the error context; *e.g.* telemetry values out of range may not matter during an observation but should be recognised during a set-up phase.

Contextual error information should be recorded as well as a simple error code and passed *en bloc* to an error logging process which is capable of making some sort of decision about the status of the error as well as recording it.

17. **The instrument-level interface**

The interface at the instrument level should be analysed and defined both in terms of protocol and transport mechanisms. Support should then be provided for different methods of access at this level, *e.g.* direct (CAMAC, RS232-C) and indirect (via DECnet, private network *etc.*). It would then be possible to design systems to cope with alternative interfaces to the outside world, thus realising the possibility of true portability for instrument D-tasks.

E.3.5 ADAM Parameter Access

1. **Data storage**

ADAM has at least three ways of storing information which is relevant to a process, *viz.* HDS, NBS and the interface file. Should these not be implemented in term of one base-level data storage scheme?

2. **Parameter list**

Parameter syntax is presently only dependent on the action – it would be more flexible to make parameters dependent on sub-actions as well.

3. **GET and SET commands**

(a) **Access to controlled instrument**

There is often a requirement to have a controlled micro perform some action when a SET or a GET is executed; *e.g.* the WHT IPCS requires many parameters to be set up before an exposure is taken. Doing multiple SETs is a natural way to proceed, but the values then never get any further than the IFL. It is often artificial to do this via a specially named action and it would be better to have an optional action (and/or subroutine) attached to the invocation of a SET or a GET.

(b) **User unit conversion action**

There should be an action (and/or subroutine) invoked before the PAR system does its range checking to provide for conversion to user units. The user requirements for the WHT system dictate that everything above the D-task level is done in user units

and that conversion is therefore done at the D-task level. In addition, the conversion need not involve a numerical equation, but might map a filter name to a slide position. A user conversion routine would take the supplied string and perform the appropriate conversion before submitting the mapped string for range checking. So, one would get:

On input:
BEGIN SET
    DO UnitConvert
    DO RangeCheck
    DO Attached action
END

In this example, each **DO** is a user-definable subroutine, or possibly an ADAM 'OBEY' action. The **UnitConvert** action converts the input user units to machine units for the range check *e.g.* filter name GG437 to filter slide position 3 (constrained to be in the range 1 to 6 say). The **RangeCheck** action is performed by the PARSYS according to the prescription in the IFL. Finally, the **Attached action** performs some device-oriented action after the range check - for example, passing the value onwards to the controlled device.

On output:
BEGIN GET
    DO Attached action
    DO UnitConvert
END

A similar arrangement also holds for **GET**. Here, the **Attached action** can cause the value returned by the parameter system to be obtained from the micro rather than the PARSYS database. The **UnitConvert** action then converts the returned value (or one from the database) into user units for return to the caller.

4. **The parameter system**

(a) **Relationship to D-tasks**

User Interface related code for accessing user supplied parameters should be moved to a higher level and the opportunity taken to simplify the interface at the lower levels. The sorting out of user-provided parameters and determining whether the user supplied an integer, a character string or an HDS file substructure is not really a function which is appropriate at the lowest level in a system.

(b) **Relationship to noticeboards**

The integration of PARSYS with noticeboards should be investigated to minimise the number of different ways of storing and accessing data. That is, find a way of replacing PARSYS by noticeboards or use noticeboards to implement PARSYS.

(c) **PARSYS functionality**

The parameter system has a number of strange features which, in the view of the author, do not contribute to a 'user-friendly' system. If an integer parameter with a defined range requirement is specified in the interface file (IFL), then:

i. Using ADAM_SEND to SET an integer value which is out of range is not faulted until the value is later recovered using GET. The faulty SET cannot therefore be easily identified. It should be faulted immediately.

ii. SETting an invalid value (*e.g.* a character string) is not faulted again until a GET, which returns the last correctly set value and an 'ADAMERR %PAR, Null parameter value', rather than '...illegal parameter value'.

iii. SETting a real value causes rounding to occur with no associated status return indicating mismatched data types.

iv. An OBEY with an invalid parameter isn't detected until the D-task ACT code attempts to GET that parameter. This means that an action has been started which should not have been and, moreover, the D-task author is forced to detect the possibility of error (a service which should be provided by PARSYS) and CANCEL the error. If the ACT code doesn't account for such possibilities, then the process could hang up and bring down the entire system.

(d) **Variable Handling**

The parameter system handles variables differently according to the task type; *e.g.* CD-tasks lose variable values after an action is executed, whereas D-tasks do not. It would be better to decide this on a per-variable basis in the interface file rather than globally on a per process type basis. This comment is related to the one regarding the proliferation of ADAM process types.

5. **PARSYS complexity**

Do **individual** processes really need to have a complex parameter system if the user interface is properly designed?

The need to have such a complex parameter system at the D-task level should be examined. If there is no need for user access to D-tasks other than via a user interface, then the complexity is not required. The corollary to this is that direct user access to D-tasks should be restricted so that the interface can be simplified. In most cases, D-tasks are driven by command procedures or CD-tasks, which can be programmed to provide a consistent set of simple parameters.

Maybe, in order to allow test harnesses to be written, ICL should be improved to cope with this area and thus obviate the need to run processes directly from DCL.

6. **PARSYS User-Friendliness**

At present, PARSYS cannot be termed 'user-friendly'. For example, in the existing INT (Perkin-Elmer) FITSOUT program, the user is asked to name the tape deck to which the FITS tape is to be written and is offered a default along the lines of `MAGO:`. The program will, however, accept `MAGO`, `MAGO:` and `1`. For the VAX equivalent using the parameter system, there seem to be problems, since MAG_ routines only accept parameter values entered into the DEVDATASET database, meaning that extra entries have to be made in order to allow the colon to be optional. As for just allowing a minimal unique description such as 0, informing the user that 0 isn't unique (say MTA0 and MTB0 both exist), or even informing the user of the legal choice on a given machine, this would seem to be impossible. However, not to do so is retrogressive given what is presently available to, or can be reasonably expected by, a user.

7. **Support for unit conversion**

There should be defined support for the conversion of units supplied to a task (D-task or whatever) into units supplied to the device or used in the task output. This extends to numerical conversion by polynomial (*e.g.* central wavelength to grating angle conversion) where coefficients need to be provided, maybe as arrays of numbers, and to database

lookup (*e.g.* conversion of a filter name into a table of related information such as position in filter slide, central wavelength, neutral density factor *etc.*).

8. **Data array transmission**

   ADAM has no support for transmitting small or medium size data arrays between processes if these exceed the size of an ADAM message. For example, the WHT system would benefit from being able to send the IPCS S-distortion correction array ($\simeq$ 512 elements) directly to the IPCS D-task for downloading into the IPCS hardware. Use of queued data blocks plus an AST mechanism would help here – a pool of medium size data blocks which can be removed and filled by one process for sending to another is already successfully used within the WHT control system.

E.3.6 User Interfaces and Command Languages

1. **General**

   ADAM as an entity doesn't always act as the user expects; there are quirks such as the necessity to capitalise the text of a direct D-task SEND which make it difficult to use. It is vital to consider how the user will behave, not how the implementor wants him or her to behave. Connected with this is the necessity to have good help and error reporting together with facilities which allow an equitable and convenient (to the user) escape from a problem or error situation. A consistency of approach, presentation and functionality without too many apparently *ad hoc* ways of doing the same thing would dramatically improve the usability of the ADAM system and reduce the scope for user confusion.

2. **Command Language Syntax**

   ICL supports an ADAMCL-style syntax for good reasons of compatibility. However, the trend should be towards a DCL-like syntax and seamless merging into VMS. Users coming to most telescopes will be adept at using DCL in the normal course of events and should not have to learn a new approach for a short observing period. For daily analysis work, it would then also complement VMS. I suspect that few astronomers write ADAMCL/ICL procedures – that this is so is borne out in the Perkin-Elmer INT system. Further proof of the validity of retaining a VMS style is offered by the existing Isaac Newton Group telescope control system interfaces, all of which adopt a DCL approach and which have been highly commended by observers.

   The need for ADAM tasks to be executable independently of, say ICL, within a DCL context should be reviewed. This is especially true given the features provided by U-tasks, which have a built-in command interpreter and user interface.

3. **ADAM process names**

   An attached command language should not need to report process names to the user – for example, 'Loading ISIS into 0A00ISIS'. To do so is an admission that the system fails to handle processes properly in the case of error. See also the section on exit handlers.

4. **Case insensitivity**

   ADAM violates the accepted Starlink standard regarding case insensitivity of user input. Direct SENDs to D-tasks are obliged to be in upper case, otherwise they don't work. To avoid additional overheads within the D-task kernel, this problem should be fixed at the command language interface level.

5. **Problems with ICL**

The following set of problems found during the recent commissioning of the WHT Faint Object Spectrograph system has been reported to Jeremy Bailey.

(a) **Directories**
ICL resets the process default directory so that, on exit from ICL, one can end up in a different directory. The original process default directory should be restored on exit from ICL.

(b) **Error Handling**
It isn't possible to discover the reason for an error from within an ICL procedure; it isn't therefore possible properly to recover from errors.

(c) **Existing D-tasks**
ICL should be able to KILL D-tasks existing in the system when it is loaded. ADAM-CL already offers this capability. Alternatively, processes should be guaranteed to disappear when the (parent) command language which loaded them exits.

(d) **Delayed actions**
ICL waits unnecessarily before performing certain actions. A call to LOAD a .ICL file from within a procedure is not executed immediately but is 'queued' until the procedure ends. Furthermore, if a series of such LOADs is written into a procedure, only the last LOAD is performed.

(e) **File location**
Because ICL resets the process default directory when a DEFAULT command is issued, it is possible to populate directories with copies of the SAVE.ICL file on exit. The location of this file should be specified by logical name.

(f) **Reserved words**
ICL reserves a number of keywords for its own use. If these are accidentally used as procedure names, the action required to recover the correct use of the keyword is undocumented. ICL should either forbid the use of such keywords as procedure names or provide a properly documented recovery mechanism.

(g) **Exit handler**
It would be useful to be able to specify an ICL procedure to be executed on EXIT in order to perform any tidying up.

(h) **Documentation**
The documentation no longer adequately reflects the facilities which are available. For example, ICL is supposed to have had a callable interface since V1.4, but no mention of this is made in the documentation.

(i) **LOADW**
The ICL LOADW command should utilise a termination mailbox to discover when a process dies or is killed.

E.3.7 Portability

1. **Architectural Portability**

Some thought has been given within the ADAM community to a version which would run under another operating system; Unix, for example. It must be recognised that such

requirements (deriving largely from considerations of hardware rather than software costs) are likely to compromise instrument control applications.

That this can happen is becoming obvious from the difficulty in adapting the existing VAX version of ADAM to work with the WHT Control File system, the WHT Utility Network and so on. Very careful consideration indeed must be given to the need to port ADAM to any other (*e.g.* Unix) type of machine. My impression is that porting Perkin-Elmer ADAM to the VAX resulted in M-tasks being devised to address the problem of process creation speed under VMS. Direct porting to other architectures without careful consideration of the consequences must be avoided. Consideration of language compatibility is just not sufficient.

Any such port would also lead to difficulties in supporting two versions of ADAM, given current financial and manpower considerations, and could be an unfortunate step particularly if, as is being rumoured in the computer press, VMS is being rewritten with portability (especially to Intel 80386 processors) in mind.

2. **Site Portability**

   Portability of control and analysis processes between sites has been a much vaunted advantage of using the ADAM system. However, this has not happened to any great extent. In any case the scope for porting control processes is less than it is for analysis processes simply because a control process has to work within a more complex environment. A N Other's D-task will work elsewhere only if the hardware is connected in the same way and the complete interface to the user is the same. No existing D-tasks will, for example, connect directly into the WHT system where the following constraints apply to what one might call a 'common-user' system:

   (a) It must use the Utility Network for commands and status.

   (b) It must use the Detector Memory System for handling image data.

   (c) The D-task must support noticeboards and update status on them in order to work with the Control File system.

   (d) Certain standard D-task commands are expected to be provided, such as initialise (warm-start).

E.3.8 The Noticeboard System

The noticeboard system (NBS) is one of the central and most valuable parts of the ADAM system in the WHT implementation. Nevertheless, it would benefit from additional functionality in the following areas:

1. Support for building and accessing noticeboards within the D-task context. It is not sufficient to expect a D-task writer to organise special code for producing his noticeboard. RGO have addressed this by writing a utility to build so-called (binary) 'noticeboard description files' from an ASCII description file.

2. General support for noticeboards must be made available at all levels within the ADAM system. ICL should support a variable type which corresponds to a mapped noticeboard variable, *i.e.* the variable is pre-declared as mapped to a particular location in a noticeboard. It appears that the latest version of ICL offers functions to GET and PUT noticeboard variables which solves the problem, but not (in the author's view) in the most elegant fashion.

3. Support for arrays and structures. Some is already available, but needs to be enhanced in the direction of array element access by name. This is required to address problems in handling lists of values describing, for example, the contents of a filter slide.

4. Support should be provided for 'aliasing', that is the ability to provide alternative ways of accessing data in the noticeboard. It might be that a process would require access to a single data item which is embedded within an otherwise complicated structure. Currently, any such access would require knowledge of the structure containing the data item; it should be possible to nominate direct access to the item when building the noticeboard.

5. It seems to be necessary to call NBS_PUT_VALUE to write a value into a noticeboard before another task can get a locator to it, despite having set up the structure. This means the initialisation phase of a process must write values into its noticeboard before anyone else tries to get locators during their initialisation phases (*e.g.* the Control File Executor).

6. It seems that the current versions of NBS allocate noticeboard memory as byte strings of the required length. This means that the start address of a string is not guaranteed to lie on any particular boundary, *e.g.* word, longword, quadword. For some purposes this is not acceptable; I/O may require word aligned buffers, and the LIB$ queue handling routines require the queue header to be quadword aligned.

7. If a process which is writing to a noticeboard by the 'correct' method happens to crash or to hang, then access to that variable by another process may no longer be possible using standard noticeboard system routines. This problem is known about, but no solution has been provided at the present time.

E.3.9 General points

1. Since data archival is a major feature of all ADAM observing systems shouldn't this be supported as a standard feature?

2. Support should be provided for mimic-type displays. For the WHT ADAM system, a single data-driven mimic process has been written to be applicable to any instrument or detector and to be easily reconfigurable. Access to D-task data is via noticeboards and updates are driven by AST action from the D-tasks.

3. It should be standard practice to use facilities provided by VAX/VMS wherever possible.

4. Greater emphasis should be placed on disseminating ideas for new features and modifications within ADAM. Too often, it seems that such 'upgrades' are the result of a *fait accompli*.

5. Given the paucity of effort available for ADAM support, is it not important that the use of such effort is maximised in the direction of user requirements, in addition to what the ADAM programming community perceives as necessary? A formal mechanism must be found to match the future development of ADAM to the evolving needs of its users.

6. ADAM is presently trying to be all things to all men. In the days when ROE were developing ADAM to run under VAX/VMS for system control at UKIRT (and use at JCMT), understandably emphasis was given to those aspects most important for controlling an on-line system. Now, as much if not more emphasis is given to the requirements of analysis applications. If we are to retain ADAM as a single environment then equal weight must be given to both aspects.

E.4 Epilogue

ADAM really needs a complete rethink after careful consideration of its architecture based on an analysis of the widening range of user requirements and experience which it is now expected to address. As part of such a re-examination, the components and layers of a substantial system (hypothetical if necessary, but preferably a real system in current use) should be analysed to discover the functionality required now and in the future. The overall direction which ADAM takes is insufficiently well defined (opening the way to implementational anarchy), so planning is needed at this level too. New and existing facilities within ADAM must also be supported and integrated in a consistent fashion – support cannot be allowed to lapse nor can it be allowed to be done in an *ad hoc* way.

At the last ADAM workshop (Hawaii, September 1987), a number of problems of duplication in certain functional areas were successfully resolved. This work needs to be continued and expanded as a matter of urgency, otherwise ADAM is in grave danger of being rejected for future telescope projects. Given that ADAM was originally conceived as a means of making better use of limited resources, some of those resources now need to be applied to putting ADAM on a better footing for the future as a package which **does** satisfy its users.

If ADAM is worth retaining then it should be carefully looked after.

E.5 Acknowledgements

This document has resulted from comments and representations made by members of the La Palma Software Group, especially Jonathan Burch and Duncan Muir, who have assiduously tracked down and solved some of the more obscure features and bugs referred to above.

I would like to thank Charles Jenkins, Rachael Padman, Mick Johnson and members of the La Palma Software Group for their comments on this report.

Part IV

Proposals

Appendix F

Proposal for MAX and MIN Parameter Responses

Submitted by: William Lupton, AAO, 6th July 1989.

F.1 The Problem

The Figaro parameter system permits responses of 'MAX' and 'MIN' when obtaining numeric values. 'MAX' results in the maximum allowed value being returned and 'MIN' results in the minimum allowed value being returned. The appropriate values are specified in the argument list to the parameter reading routine.

The ADAM parameter system does not support these responses. It is necessary to do so in order to be able to provide the user interface in the ADAM Figaro monolith that Figaro users will expect.

F.2 Proposed Solution

1. The response 'MAX' or 'MIN' (case independent) should be permitted for parameters obtained using PAR_GET0x where x is in (I,R,D).
2. New routines:
   PAR_MAX0x(PARAM,MAX,STATUS) and PAR_MIN0x(PARAM,MIN,STATUS)
   where x is in (I,R,D,C) will be provided. These will have the effect of adjusting the relevant limit for the parameter, with the proviso that if a range has been specified in the interface file then these routines cannot set a limit that is outside the original interface file limit.
3. 'MAX' will cause the upper limit to be returned and 'MIN' will cause the lower limit to be returned.
4. If a 'MAX' or 'MIN' response is given and there is no corresponding limit then an appropriate message will be output ('maximum value not known' and 'minimum value not known' perhaps) and the user will be re-prompted.
5. This proposal does not provide a means of returning a maximum or minimum character value (although it does provide a means of setting character limits). Is such a method necessary? Figaro does not provide one.

Appendix G

Proposal for Multiline HELP on Parameter Prompts

Submitted by: Jeremy Bailey, JAC, 22nd August 1989.

G.1 The Problem

ADAM allows a '?' response to a parameter prompt, which results in a single line of help text, coming from the HELP field of the parameter description, being output on the terminal. A single line of text is often inadequate to give useful information on the parameter. Figaro, for example, allows multiple line HELP information under the same circumstances.

The size of the help text cannot easily be increased. This information is stored in the parameter system common blocks, which would have to be increased in size substantially. Also the help text, along with other information, is passed to the user interface as a message, the size of which is limited.

G.2 Proposed Solution

The proposed solution is to use the HELP field in the interface file to contain the name of a VMS HELP library, and one or more HELP keys. Fields of this type would be distinguished from normal help lines by a suitable initial escape character. I propose using the % sign for this purpose. When the user interface has a help field of this form, and the user responds with a '?', the user interface will retrieve the corresponding text from the HELP library and display it.

For example, an interface file entry of the form:

```
HELP '%KAPPA_DIR:KAPPA KEY1 KEY2'
```

would result in the help text being retrieved from KAPPA_DIR:KAPPA.HLB using keys KEY1 and KEY2.

I have already implemented the change in ICL (actually in the UFACE routines used by ICL) without any difficulties. To be generally available, the facility needs to be included in all user interfaces. SMS may present more problems.

Editor's Note: The implementation of this proposal will be in ICL Version 1.5. However, discussion on the proposal, carried out in the ADAM_DEVELOPMENT VAX Notes conference, has suggested some modifications.

Appendix H

ICL Enhancements

Submitted by: Jeremy Bailey, JAC, 22nd August 1989.

The following are extracts from the ICL Version 1.5 Release Notes. They describe the enhancements made to ICL in response to actions placed at the 1989 ADAM Workshop.

H.1 Real Variables Now in Double Precision

All real variables are now represented in double precision (about 16 decimal digits of precision).

In order to ensure upward compatibility, the default behaviour when performing unformatted conversions of real variables to strings is to convert with six digits of precision as in the old version. This ensures that the resulting strings have the same size as previously.

Full precision conversion to strings can be obtained using formatted conversion (with the colon operator) or by controlling the precision of unformatted conversions using a new command SET PRECISION n. Here n is the number of digits of precision and can range from 1 to 16.

H.2 SAVEINPUT Command

This command has the form:

```
SAVEINPUT n filename
```

and causes the last n lines of input to be saved in the file with name filename. If filename is omitted the file SAVEINPUT.ICL in the default directory is used. If n is omitted the entire contents of the input buffer are saved.

H.3 $ Symbol in Place of DCL Command

The $ symbol, if it occurs as the first non-blank character on a line is interpreted as equivalent to the DCL command. Thus:

```
$TYPE SAVEINPUT.ICL
```

is equivalent to:

```
DCL TYPE SAVEINPUT.ICL
```

H.4 Removal of DIR

The DIR command has been removed from the language. It was something of an anomaly as it was the only DCL command provided in this way. The DIR command was exactly equivalent to $DIR or DCL DIR, and can therefore be provided if required by including the line:

```
DEFSTRING DIR DCL DIR
```

in the LOGIN.ICL file.

H.5 New Functions

H.5.1 The ELEMENT Function

The ELEMENT function provides the equivalent of the F$ELEMENT lexical function in DCL.

```
ELEMENT(n,delim,string)
```

returns the nth element of <string> where string is divided into elements using the delimiter character <delim>. If the element does not exist it returns the delimiter character. The first element in the string is numbered zero.

For example:

```
ELEMENT(3,'/','Mon/Tue/Wed/Thu/Fri/Sat/Sun')
```

returns 'Thu'.

H.5.2 The FILE_EXISTS Function

The function FILE_EXISTS has the form:

```
FILE_EXISTS(filename)
```

and is a logical function returning TRUE if the file of name <filename> exists.

H.6 Removal of Callable Figaro

Previous versions of ICL included direct access to callable Figaro. This has now been removed. Equivalent behaviour can be obtained by defining callable Figaro commands using the DEFUSER facility in the form:

```
DEFUSER SPLOT BIGFIG FIGARO
```

where SPLOT is the command being defined, BIGFIG is the logical name for the callable Figaro shareable image, and FIGARO is the subroutine to be called within that image.

Appendix I

ADAM Related VAX Notes Conferences

Submitted by: Alan Chipperfield, RAL, 7th September 1989.

This report is derived from the text of a mail message distributed via ADAMSC on 7th September 1989.

I.1 The ADAM_DEVELOPMENT Conference

Following a decision at the 1989 ADAM Workshop, a public VAX Notes conference has been set up for discussion of ADAM system development issues and dissemination of news on ADAM developments and forthcoming releases.

The conference will be used initially to discuss actions, proposals and conclusions arising from the Workshop. However, because almost any comment could give rise to a system development, it is admissible for anyone to create a topic on any ADAM-related subject other than bug reports, which should be mailed to RLVAD::STAR (see below).

> The ADAM Development Conference is RLSTAR::ADAM_DEVELOPMENT.
> The moderator is Alan Chipperfield (RLVAD::AJC).

I.2 The BUG_REPORTS Conference

Furthermore, again following agreement at the Workshop, all ADAM bugs should now be reported to RLVAD::STAR. A conference has been set up to enable anyone to follow progress of a bug report about any Starlink software item, including ADAM.

> The Starlink Software Bugs Conference is RLSTAR::BUG_REPORTS.
> The moderator is Dave Rawlinson (RLVAD::STAR).

Appendix J

Proposal for Interface File Search Path

Submitted by: Alan Chipperfield, RAL, 12th October 1989.

J.1 The Problem

A user requiring a private version of an interface file has to copy the executable image of the task to the same directory as the private interface file.

It was agreed by the 1989 ADAM Workshop that it would be useful if interface files were located by looking along a search path ADAM_IFL similar to, but distinct from, the ADAM_EXE search path used to locate executable images.

It was recognised that this feature only becomes really useful if there was an 'include' feature in interface files so that private versions of individual bits within monolith interface files could be easily used. This aspect of the problem has not yet been addressed.

J.2 The Proposal

When a task is activated, the required interface file is found by having SUBPAR_ACTIV, SUBPAR_ACTDCL or SUBPAR_ACTSHR call a new routine SUBPAR_FINDIF. The new routine attempts to translate the logical name ADAM_IFL using the tables defined by the SYSTEM logical name LNM$FILE_DEV. (Normally PROCESS, JOB, GROUP and SYSTEM in that order.)

ADAM_IFL may be a search path and, if it is defined, it is used as the filespec, together with a default filespec of *filename*.IF% (where *filename* is the filename of the executable image and % is any single character), in a call to LIB$FIND_FILE. If such a file is found, the type is checked. If the type is not .IFC or .IFL, the routine continues searching until no more files matching the specification are found.

If ADAM_IFL is not defined, or an interface module is not found using it, the routine attempts to find the interface module in the same directory as the executable image.

If an interface module is found, the common blocks are set up using SUBPAR_LOADIFC (if the filetype is .IFC) or PARSECON_READIFL (if the filetype is .IFL). If the routine is unsuccessful, a message is typed and an appropriate STATUS value is returned.

Notes:

1. ADAM_IFL will be entirely the responsibility of the user. No default value will be defined in the standard system.

2. It is recommended that ADAM_IFL be defined as a JOB logical name only in relevant sessions. It cannot be a PROCESS logical name if the task is to be run in a sub-process, and GROUP logical names may cause confusion as they remain set between sessions. There will be a slight overhead in having ADAM_IFL set when it is not required.

3. If ADAM_IFL includes filename(s), the name of the executable image will be overridden.

Appendix K

Thoughts About a Distributed Version of NBS

Submitted by: William Lupton, AAO, 12th December 1989.

This report arose from an out-of-hours discussion at the Workshop between WFL, JAB, BVM, BDK, JMB and LRJ. Where there are areas that I believe require further thought, I have bracketed a remark within '????' symbols.

K.1 Introduction

ADAM processes communicate using the message system to send messages and the noticeboard system to share memory. From ADAM version 2.0 the message system will support the sending of messages to tasks on other machines provided that the machines are connected using DECnet. However there are currently no plans to permit such tasks to share data using the noticeboard system.

At the ADAM Workshop there was a discussion of possible applications of a noticeboard system capable of operating over networks. It was generally agreed that such a system would be useful and feasible and there was some discussion of its implementation. The purpose of this note is to summarise the discussion and to provide a concrete proposal from which to proceed.

K.2 Applications

The major perceived application of a distributed noticeboard system is to permit a single ADAM system to exist on several machines. Initially, such machines will have to be connected using DECnet.

Another potential application would become viable if the ADAM message system became supported over networks other than DECnet. For example, it might be supported over the La Palma Utility Network, over a simple point-to-point network as envisaged by the AAO, or over a VAX to transputer network as envisaged by JAC. Assuming that the same network connections were used in implementing the distributed noticeboard system, it would then be possible to put a value into a noticeboard on a transputer and for that value automatically to appear in a noticeboard on a VAX.

K.3 Proposals

Consider Figure K.1. Two tasks T1 and T2 reside on processor P1; T1 owns noticeboard N1 and T2 is reading it.

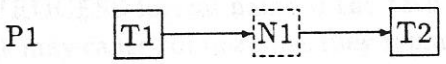

Figure K.1: Single Processor.

This is the normal current situation. Each of T1 and T2 can make normal noticeboard system calls and everything works fine.

Now assume that T2 resides on processor P2 (Figure K.2). It cannot read noticeboard N1 because there is no noticeboard N1 on processor P2. But what if there were and there were some magic way of forcing P2's N1 to have the same contents as P1's N1? Let us refer to such 'magic' tasks as M1 and M2. M1 and M2 should be identical.

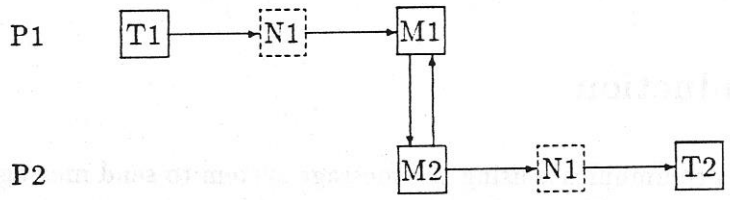

Figure K.2: Double Processor.

T2 could now run normally on P2 and would not have to know that its friend T1 is running on a different machine.

How can this be achieved without completely swamping whichever network connects P1 and P2? There are two basic techniques that can be employed:

1. T1 writes to N1 and somehow notifies M1 that P2's copy must be updated. M1 notes the new value, sends a message to M2, and M2 updates P2's copy of N1.

2. T1 writes to N1. M1 notices that the value has changed and updates it in the same way as above.

Technique 1 has the advantage that an important item will be updated immediately and has the disadvantage that every noticeboard update will require a message flow across the network. One of the reasons for using the noticeboard system in the first place is to avoid such message flows.

Technique 2 has the advantage that several updates can be batched into a single message. Also, the polling rate can be high for 'important' items and can be low for 'unimportant' items. It has the disadvantage that CPU time will be used even when no noticeboard activity is taking

place. Also, it will not be acceptable to poll even 'important' items fast enough, since to do so would inevitably cause unacceptably high usage of CPU time.

It is clearly necessary to employ a mixture of the two techniques. Somebody will have to provide the following information for each noticeboard item that is to be maintained on P2:

1. Whether a change to the item is to be notified to M1.
2. The polling interval for this item. ???? only if changes are not be notified ????

K.4 Problems

The proposals outlined in the previous section will permit the important parts of a noticeboard to be maintained on another processor and will guarantee that resources are used for maintenance of items that are regarded as being important. There are still some problems though.

Firstly, the above scheme does not guarantee that items will be updated in P2's copy of N1 in the same order as they are updated in P1's copy. When all tasks are in the same processor this is of course automatically the case and programs have quite rightly been written to make that assumption. It is therefore necessary to provide a call that T2 can make which will result in a message to M2 asking it to guarantee that P2's copy of N1 corresponds to P1's copy. If T2 is to be unaware whether it is accessing a distributed noticeboard, it must be able to make this call without ill-effect in the case where T1 and T2 are both on the same processor.

Secondly, ???? there must be some other problems ????

K.5 Implementation

Given that the implementation of this scheme requires messages to be sent between processors, it makes obvious sense for the M1 and M2 tasks to make use of the ADAMNET process that will be available from ADAM V2.0. For efficiency reasons (to save unnecessary messages) it will probably make sense to implement M1 and M2 actually within ADAMNET but this is not strictly necessary.

We will assume that T1 is loaded and N1 is created before T2 is loaded (this is normal practice even when T1 and T2 are on the same processor). It is necessary to arrange for P2's copy of N1 to be created before T2 can be loaded and this must be done either by direct contact with M1 on P1 or with M2 on P2. It doesn't really matter from which end this communication takes place and indeed it may be necessary or convenient to permit it from either end. However, for the sake of the discussion we will assume that it takes place on P2. A utility program on P2 will talk to M2 saying:

1. Please create a local copy of processor P1's noticeboard N1.
2. I am interested in items A.B, C.D, ... and I wish them to be polled with frequency x, y, ... seconds. ???? Allow wild-carding as in A.* or A... ????

M2 will pass this information to M1, which will find the requested noticeboard and will pass the definition part of it back to M2 (???? pass contents of definition file? pass whole noticeboard? new noticeboard system facility ????). It will then update its data structures with the information about the items in which T2 is interested. In the general case, more than one task

may be interested in a given item. Finally, it will send the current values of those items back to M2. Meanwhile, M2 will create the noticeboard (if it does not already exist) and will plug the received values into the appropriate places.

M1 will now poll the modified counts for the relevant items at the appropriate rates. When it detects that an item has changed, it will read its value and will send it to M2. M2 will set the value in P2's copy of N1. Note that M1 cannot simply send an ID across the network since IDs are in fact virtual addresses. It would be possible to provide a transportable ID by subtracting the base address of the noticeboard (a new noticeboard system facility would be required) and this, together with the noticeboard name and the new value, would be sufficient information to pass to M2. To pass the name of the item would be less efficient but I suppose that names could be cached and it might not be too bad.

If a given polling cycle detects that multiple items have altered, it should be possible to batch multiple updates into single messages between M1 and M2. There will be an upper limit to the length of these messages and where an item is over a certain size a single update may have to be split between multiple messages.

If T1 wants to arrange for immediate update of a given item in P2's copy of the noticeboard, it will have to call a new routine saying 'when I update this item I want M1 to be notified'. T1 will continue to use standard facilities actually to update the item, but clearly there needs to be a new low-level noticeboard facility to support this. It is possible that T1 will call this new routine when M1 does not exist (in which case no attempt will be made to notify M1 on the update of such items) or before M2 has told M1 that it is interested in noticeboard N1 (in which case M1 will ignore update messages until it knows that somebody is interested in the item in question).

If T2 wants to ensure that it has a consistent copy of N1, it will have to call a new routine saying 'guarantee me that all updates to items in which I am interested have been reflected in my copy of the noticeboard'. This routine will send a message to M2, which will send a message to M1, which will immediately poll the modified counts of all items in which T2 is interested and will send modified values back to M2 and thus to P2's copy of N1. M1 will have to check that the overall noticeboard modified count does not change whilst it is doing this since if it did change it is possible that what was sent to M2 would not represent a snapshot of the noticeboard state. In this case, the poll of the modified counts will have to be repeated. ???? This could be a problem if the noticeboard is rapidly being updated. May need several flavours of the routine, one of which cares less about complete consistency. Also may need to be able to disable further updates across the network until another routine enables them, since otherwise, although P2's copy of N1 may instantaneously be valid, it could immediately become invalid ????

K.6 Changes to Existing Software

Some changes will be required to the noticeboard system. Others may be desirable. I believe that only the first one is strictly necessary. The routine names and argument lists are not definite proposals:

1. NBS_PUT_TRIGGER (ID,ROUTINE,STATUS)
   This routine specifies that whenever the item with ID is updated (by NBS_PUT_VALUE or NBS_PUT_SHAPE) routine ROUTINE should be called (in the context of the process of the updater): CALL ROUTINE (ID,STATUS). This facility can be used to implement the trigger on update of an item. ROUTINE will convert ID to the noticeboard name and a transportable ID and will send them to M1.

2. NBS_GET_MODIFIED_POINTER (ID,POINTER,STATUS)
   This routine returns a pointer to the modified count of an item or noticeboard. Use of such pointers will allow very rapid checking of which items have altered.

3. NBS_GET_UPDATED (ID,UPDATED,STATUS)
   This routine determines whether the noticeboard or an item has been updated (using NBS_PUT_VALUE or NBS_PUT_SHAPE) since the last time that this process called this routine on behalf of this item. This achieves the same effect as can be achieved using NBS_GET_MODIFIED and NBS_GET_MODIFIED_POINTER but in a more 'official' way.