Practical experience with ALGOL 68-RT

J. R. Oliver* and R. S. Newton†

The use of ALGOL 68 to give all the facilities of an online, multi-access system has many advantages. This paper outlines the use of ALGOL 68-RT to provide parallel processing features in an ALGOL 68-R program and describes the practical benefits that military users at HQ RAF Strike Command have obtained.

(Received January 1978; Revised February 1979)

One of the computer facilities needed by HQ RAF Strike Command for its ICL 1904S* was an online, multi-access computer facility which used a fully validated, structured data base for information storage and retrieval and for a combination of mathematical computation and data processing. There had to be interrogation facilities to extract records according to multiple criteria and an enquiry facility to resolve certain complex mathematical relationships between the records in the system. There was the need to provide multi-threading so that different users could use the same service concurrently, and the need to enforce single-threading to restrict a service to only one user at a time when the program overheads of providing multi-threading were too high. It required software that allowed a large program to be written quickly with high reliability—given certain hardware constraints, provide growth potential in a large program, and allow maximum flexibility to change. The software chosen was the ALGOL 68-RT Online System.

The ALGOL 68-RT online system

The ALGOL 68-RT Online System (Bond, 1970; Newton *et al.*, 1976; Woodward and Bond, 1974) has been developed under the leadership of Dr D. P. Jenkins by the Computing Group at RSRE Malvern and has now been used by HQ RAF Strike Command for four years. This is undoubtedly the most powerful software system of its type available for an ICL 1900. It offers multi-access operation on visual display units (VDUs) with the entire application coded in ALGOL 68-R as a single program with the systems functions relatively isolated from the applications programmer. For multi-threading functions, code is re-entrant and there are protection mechanisms to prevent the inputting or amendment of the same record at the same time by different users.

The online system functions are provided as a procedural package and perform two roles. Firstly, there are a set of procedures which are called to define the hardware and software configuration for a particular run. Secondly, procedures are provided for the control of and interaction with the input/output devices.

Processes (Wilkes, 1975)

The operator at his VDU will see the program as a set of sequential functions which will perform his current job. The logical execution of this sequence of functions by the processor is called a 'process'. Operators at the other VDUs will have a similar view of the program. Thus, at run time, the program will consist of many processes, one per VDU plus several dedicated to the online system itself, each conceptually executing independently of the others, but communicating where necessary in a controlled way.

*HQ RAF Strike Command, High Wycombe, Bucks HP14 4UE
†Royal Signals and Radar Establishment, Malvern, Worc. WR14 3PS

Organisation of processes (Woodward, 1973; Woodward and Bond, 1974)

Given this definition of a process, **Fig. 1** shows the ALGOL 68-RT concept on which the online system is built. The outer loop shows how control is passed from user process to user process by the 'co-ordinator' (or scheduler) which decides which process to run next. Each of the inner loops indicates how each process can consist of several sequential functions. The mechanism to achieve this is an additional library for the ALGOL 68-R system; no alterations were made to the ALGOL 68-R compiler. The basis of ALGOL 68-RT is a language construct which will launch a procedure as an independent process. ALGOL 68-RT runs an ALGOL 68-R program as a series of stacks which are set up by the syntax-directed compiler, each procedure forming an independent stack in the computer. Launching a process puts this process into a ring which is examined by a 'round-robin' scheduler. Effectively, the VDU user owns an RT process (Newton *et al.*, 1976)—written as a user procedure—that is, from time to time, halted and reactivated by the 'co-ordinator'. This process controls the man/machine dialogue at a VDU and the procedure is used re-entrantly to control as many VDUs as are linked to the system. This 'user procedure' is structured as a hierarchy of procedure calls, each representing a particular function. The diagram shows the cyclic nature of each process, and the time-slicing between them. Transfer of control between processes takes place on three occasions:

(*a*) when a peripheral transfer takes place

(*b*) when the co-ordinator is called explicitly

(*c*) when an operation on a 'semaphore' calls the co-ordinator.

The scheduler also controls the overlay package. The overlay package is very powerful and allows a great variety of overlay configurations to be in core at any instant as defined by the syntax of an overlay specification.

1. *Setting the online system going*

The application is compiled and started as a normal GEORGE 'Job'. On entry to the main body of the program, the following procedures are called to define the multi-process structure and the hardware configuration and then to set the whole system running:

proc *define configuration* = ([ ] **int** *description of devices*)

This procedure sets up the necessary data structures to define the configuration. 'Description of devices' defines the number of devices to be driven and their device type, such as VDU or teletype.

proc *start configuration* = **void**

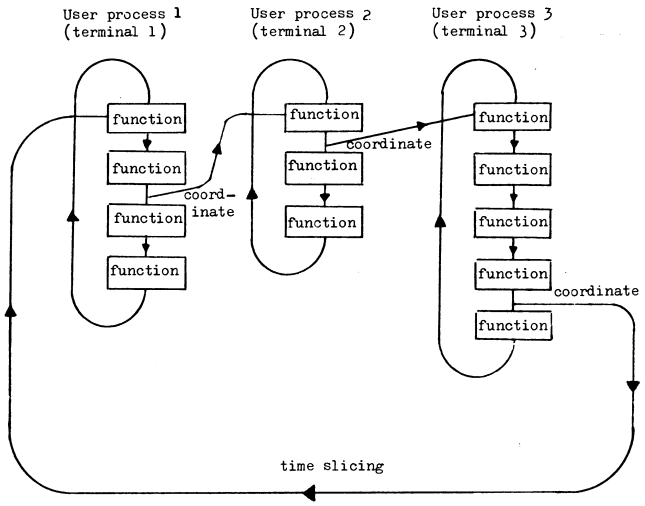

Fig. 1 The ALGOL 68-RT concept

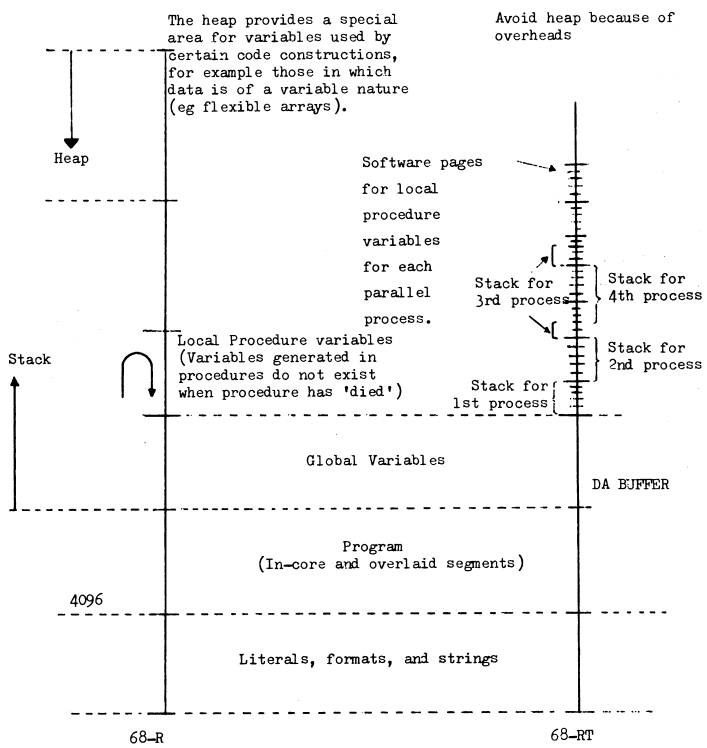

Fig. 2 Core mapping in ALGOL 68-R and ALGOL 68-RT

This procedure performs initialisation of the communications hardware and its driving software.

proc *start iden* = (**int** *identifier*,
proc (ref charput, ref charput) *user procedure*)

This procedure launches the given 'user procedure' as a process, and identifies through the parameter 'identifier' which device data structure it is to use.

proc *scan configuration* = **void**

This procedure starts the system running.

Each operator may now connect his VDU to the appropriate process using 'identifier' as a parameter to the GEORGE command 'Attach'.

Core Mapping

Fig. 2 shows a comparison of the core mapping arrangements for ALGOL 68-R and ALGOL 68-RT. The major addition in ALGOL 68-RT is the division of the stack containing local procedure variables into software pages providing stacks for each process. Each parallel process has its own local storage space, normally 2 'pages' of 512 words each, although in this application a page size of 600 words was necessary. The significance of a page is that no single local object, e.g. array, can be bigger than a page. Paging overheads incur about 40 words/page. The heap was not used in this program to avoid time and space overheads.

2. *Picture generation*

One of the unique features of the online system is the powerful facility provided to aid the generation of output to a VDU and the validation of the resulting operator response. The screenful of information for the VDU is described by the use of standard ALGOL 68-R 'formats'. These allow a concise description of text together with any numerical information required from the program's run time data. ICL VDUs provide the facility for defining 'unprotected' and 'protected' areas of the screen, into which the operator can and cannot write respectively. An 'unprotected' area is represented symbolically in the formats and is associated with a validation routine.

ICL 1900 programs can exhaust LOWER data. This is due to the architecture of the 1900 whose address structure was originally designed for a 32K machine and the construction of early compilers which were designed to compile programs no greater than 32K. It is therefore imperative to have ALGOL 68-R formats, which would normally reside in LOWER, on backing storage. This is accomplished by inputting a format into the auxiliary program FRAMEMANAGER for translation into low level form on disc file, for later use by the online program. The ability to code VDU pictures directly in ALGOL 68-R has given huge advantages because—when compared to other tedious frame formatting systems—it provides all the flexibility necessary for the evolutionary development of conversational dialogues with the users.

Input/Output

One of the most attractive and useful features of ALGOL 68-RT, from the point of view of the applications programmer, is the clean, high level language interface between the online application program and the RT system. Input/output is controlled by a very powerful procedure:

See

proc *display and validate* =
(**ref charput** *vdu in, vdu out,* .....(*a*)
proc void *output to screen,* .....(*b*)
proc (int) int *input and check* .....(*c*)
proc void *restart*) .....(*d*)

which allows the clear definition of:

(*a*) the input/output channels

(*b*) the procedure to generate the output frame

(*c*) the procedure to read and validate the input data

(*d*) the procedure to be activated if the input sequence is to be abandoned and some other action carried out.

In this online system, the validation of input data is completely integrated with the language facilities available to the applications program, giving a powerful validation capability. As shown in **Fig. 3,** this procedure is best used when embedded inside another procedure—say OUTPUT AND INPUT VDU SCREEN—which can define all of the parameters needed by DISPLAY AND VALIDATE.

Program structure

Fig. 4 shows an outline example of an ALGOL 68-RT program.

Program	*Commentary*
proc *output and input vdu screen* = (**ref charput** *vdu in, vdu out,* **ref** [ ] **char** *function*):	Procedure declaration. Input parameters are I/O channels and a 2 character variable, passed down through processes to control direction of program.
begin	
[1 :92] **char** *c format*;	Declare data space for format.
proc *call format* = **format**: (*get format* (*c format*, 51));	Procedure to read format from backing storage.
proc *restart* = **void**: ( );	Procedure defining escape action.
proc *input and check* = (**int** *i*) **int**:	Procedure to read and validate input. This procedure is called initially to pick up the number of screen parameters (3 in this case) and is then called for each screen parameter in turn. It remembers any fields that are invalid—causing field markers to flash—and will only exit from DISPLAY AND VALIDATE if all fields are correct.
begin	
case *i* + 1 **in**	
3, **c** *no of screen parameters* **c**	
variable to select the current character channel.	
get (*cc, parameter a*);	Read and validate 1st screen parameter.
validate a (*parameter a*),	
get (*cc, parameter b*);	Read and validate 2nd screen parameter.
validate b (*parameter b*),	
get (*cc, function*);	Read and validate 3rd screen parameter.
validate function (*function*)	
esac	
end;	
display and validate (*vdu in, vdu out,* **void**: (*outf* (*cc, call format,* (*x, space*)), *input and check, restart*)	Call of DISPLAY AND VALIDATE with 5 parameters.
end;	

Fig. 3 Procedure to output and input data via the VDU screen

Overlaying (Bond, 1970)

The overlay capability of ALGOL 68-R and ALGOL 68-RT is particularly elegant and powerful. The unit of overlay is the segment which can contain one or more procedures which is read into core, according to an overlay specification designed to minimise disc accesses when the program is used. The overlay specification defines one or more areas of main store. At any instant when the program is running, each area will hold just one unit of overlayed instructions. In the simplest case, a unit is one segment, but it is often convenient to group two or more segments together to form a single unit which is brought into main store by a single transfer. Each area is automatically made big enough to hold the largest unit assigned to it. In the overlay specification, the segments forming a unit are identified by their titles with + as a separator, the titles making up units in the same areas are demarcated by commas (,), the area boundaries are denoted by semicolons (;), and the complete specification is terminated by a full stop. Thus, an example of an overlay specification could be:

Specification	*Comment*
*sega*1, (*sega*2; *sega*3, *sega*4);	Area 1—can be occupied by *sega*1 or (*sega*2 and *sega*3) or (*sega*2 and *sega*4)
*segb*1, *segb*2, *segb*3 + *segb*4.	Area 2—can be occupied by one of *segb*1, *segb*2, or (*segb*3 and *segb*4)

In the Strike Command application with a large program on a non-paging ICL 1904S*, the diversity of overlay configurations that these mechanisms facilitate has proved essential.

Error recovery

This program uses the mechanisms provided to prevent many types of program failure from being catastrophic. Two of the most important of these are

proc *set process event* = (**proc void** *recover*)

which associates a label—defining response to errors—with the current process so that it may be called after a subsequent fault, and

proc *process event* = **void**

which calls the event procedure most recently associated with the current process. Throughout the Strike Command program, the error recovery procedure changes from function to function, the main differences being caused by the need to ensure that in a failure and recovery situation the global data base is left consistent.

The Strike Command application program

The Strike Command application program was structured as 41 separately compiled segments, totalling 400K words in two albums, overlaid into 17K. The total core space required by the program is 47K. There are 157 frame formats on backing storage.

1. *Structured programming*

High speed computer stores contain millions of bits stored in a monotonous sequence of consecutively numbered but other-

```
segment name with rtdriver, vduframes from
rtalbum
begin
 proc rt demo = (ref charput vdu in, vdu out):
  begin
      User dialogue which calls
      procedures like that shown in Fig. 3.
  end;
 define configuration
 ((vdu 7181, vdu 7181), 1);          present copies
                                     of this procedure
 start configuration;                at 2 terminals.
 for n to 2 do
      start iden (n, rt demo);
 scan configuration
end
finish
```

Commentary

rtdriver contains procedures for 'co-ordinator'
vduframes contains procedures to manipulate vdu screen.

Procedure parameters are input and output channels.

Procedures to start system going—see Section 1.

Fig. 4 Outline example of ALGOL 68-RT program

wise equivalent storage locations. We can only attach a meaning to such a vast amount of bits by grouping them in such a way that we can distinguish some sort of structure in the vast amount of information. This structure is *our* invention and *not* an inherent property of the equipment. It follows that it is our obligation to structure 'what is happening where' in a useful way (Dijkstra, 1972). One of the main aims in writing this program was to give it a meaningful structure because of the great benefits that derive from this. The use of ALGOL 68-R as the sole programming language (system) has greatly facilitated the application of 'structured programming' principles and has, indeed, often forced the programmer to apply these principles. Throughout the 400K of application software, GOTOs are used in only three situations; in the system software, they are used in very exceptional cases (Dijkstra, 1968).

This program has demonstrated that:

(*a*) structured programming produces a better documented program with greater readability

(*b*) a programmer can read a program segment from top to bottom without having to follow through a number of transfers of control

(*c*) structured programs are easier to debug and understand at some later date, making it easier to avoid subsequent 'structural decay'

(*d*) structured programs are easier to modify, especially when the changes are made by someone other than the original programmer, since modifications can be made to isolated blocks of code without affecting other major segments of the program.

Data structures

Mode declarations were selected to make the programming as easy as possible while allowing relatively efficient execution of code. The criteria for deciding these mode specifications were:

(*a*) they were to define the entities for direct-access records

(*b*) they were to make the code as readable and as self-documenting as possible

(*c*) subscripting was to be kept to one level if possible.

Modular program construction

In this program, modular programming is based on ALGOL 68-R procedures. A major advantage of the use of procedures as the logical unit of processing in ALGOL 68-R is the rigorous compile time checking of data modes specified for use by each procedure; this has proved to be an invaluable aid to the development of an error-free program. These procedures have been grouped into segments, the segment being the unit of overlay, and these segments have then been put into two albums. The compile time mode checking that occurs when each subsequent segment is put into the album—and on many other occasions—has also proved an invaluable aid to program development, although the computer has undoubtedly had to work hard when fundamental changes in the segment and album structure have been made.

Data files, indexes, and main records

The data defining each of the items to be held in the system has been split between an index and a main record held on GEORGE 'exofiles' (i.e. outside the filestore), the position of the main record corresponding to the position of its entry in the index. The data going into the index, apart from the identity itself, comprises—in the main—'gating' data, to facilitate retrieval on multiple keys (Yourdon, 1972). The indexes then serve a double purpose:

(*a*) to list items according to criteria (i.e. inverted record retrieval)

(*b*) to justify the retrieval of main records from direct-access storage when doing mathematical comparisons, greatly reducing disc-accesses.

Record retrieval is performed using the random access from backing store (binary transput) procedures available in ALGOL 68-R.

Program data

Two categories of program data are global:

1. *Record indexes* Protection mechanisms are incorporated to reserve positions in the record indexes at which data is being input or amended during multi-threading/multi-access use.

2. *Variables used by mathematical processes* The penalties of making the variables for solving the mathematical problems local to each process were considered to be too high because this would necessitate many procedure parameters. Consequently, these processes were made to be single-threading,

which was enforced by belt and braces:

(*a*) access to certain facilities was by a password

(*b*) semaphores, these being special variables that control the relative progress of the different parallel processes (Pagan, 1976).

All other variable data—other than certain controlling variables—is local, being generated dynamically as processes (procedures) are activated.

Programmer productivity

As some areas of industry are beginning to realise, an ALGOL 68 programmer can actually enjoy what he is doing and, as a consequence, can be very productive (one of the authors suffered under PLAN and COBOL before being released from his misery!). ALGOL 68 is a language for those who do not believe that life is infinitely long; it is for those who wish to avoid the experience of trying to drain a swamp with a teaspoon. There is no denying that (like religion?) ALGOL 68 does require some initial intellectual effort, but this is not as great as some—who have not tried—would maintain. At HQ RAF Strike Command, all of those servicemen or civilians who wanted to use the language for their programs have succeeded, sometimes with some initial assistance. The additions to ALGOL 68-R to obtain ALGOL 68-RT are not great and are totally consistent with its elegance and the need to protect the poor applications programmer, who has his own problems, particularly in any large system, from the intricacies of an online, multi-access system. The advantages of this clean interface and the relative ease with which VDU frame formats can be produced and the user dialogue programmed cannot be emphasised too much. With the RAF Strike Command system, 400K of error-free ALGOL 68-RT code was produced with three man-years of work. Perhaps some commercial DP managers should note that ALGOL 68 programmers can become obsessed with the desire to produce large, overlaid, 'error-free', 'structured' programs in short timescales!

Conclusions

The combination of ICL 1904s*, GEORGE 3, and ALGOL 68-RT has proved to be a suitable environment for the development of this online, multi-access program with its combination of mathematical and data handling activity for local or remote users.

The complete dedication to ALGOL 68-R for all of the applications program, the clean interface between system and application software, and the powerful facilities to manipulate VDU screens provided by the ALGOL 68-RT online system have made it possible to produce a large, complex program in a short timescale which has high levels of readability, self documentation, reliability, growth potential, and flexibility to change.

Acknowledgements

The authors thank Dr David Jenkins, Miss Susan Bond and Mr David Hunter and the other members of the ALGOL 68 Group at RSRE Malvern for their considerable help in the implementation of this ALGOL 68-RT system at HQ RAF Strike Command.

References

BOND, S. G. (Ed.) (1970). ALGOL 68-R Programmers Manual, RSRE Malvern.

DIJKSTRA, E. W. (1968). GO TO Statement Considered Harmful, (Letter to the Editor), *CACM*, Vol. 11, No. 3, pp. 147-148.

DIJKSTRA, E. W. (1972). Hierarchical Ordering of Sequential Processes, APIC Studies in Data Processing No. 9, *Operating System Techniques*, Academic Press.

NEWTON, R. S., HUNTER, D. W., JENKINS, D. P. and MANDER, K. C. (1976). *The ALGOL 68RT Online System*, RSRE Malvern.

PAGAN, F. G. (1976). *A Practical Guide to ALGOL 68*, John Wiley & Sons.

WILKES, M. V. (1975). *Time Sharing Computer Systems* (3rd Ed.), pp. 36-38, MacDonald and Jane's/American Elsevier Computer Monographs.

WOODWARD, P. M. (1973). *Parallel Processing and Simulation*, An Introduction to the Use of ALGOL 68-RT, RSRE Malvern.

WOODWARD, P. M. and BOND, S. G. (1974). *Ministry of Defence ALGOL 68-R Users Guide* (2nd Ed.), HMSO.

YOURDON, E. (1972). *Design of On-Line Computer Systems*, p. 285, Prentice Hall.