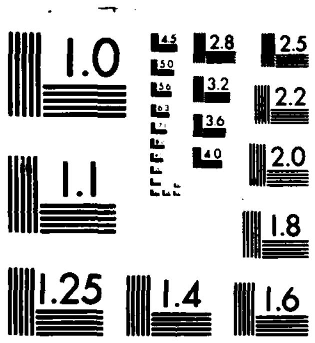
1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3

[illegible]

AF-CORAL : LANGUAGE FEATURES TO INTEGRATE MASCOT WITH CORAL 66

Author: T A D White

[illegible]

ROYAL SIGNALS AND RADAR ESTABLISHMENT

Report No 82019

Title:	AF-CORAL : LANGUAGE FEATURES TO INTEGRATE MASCOT WITH CORAL66
Author:	T A D White
Date:	November 1982

SUMMARY

This paper briefly describes AF-CORAL II, an addition to the computer programming language CORAL66, which allows a programmer to write software for a MASCOT designed system with improved clarity and program integrity.

Some knowledge of CORAL66 and MASCOT is assumed.

880/59

AF-CORAL: Language Features to integrate MASCOT with CORAL66

0. Abstract

This paper briefly describes AF-CORAL II, an addition to the computer programming language CORAL66, which allows a programmer to write software for a MASCOT designed system with improved clarity and program integrity.

Some knowledge of CORAL66 and MASCOT is assumed.

1. Introduction

MASCOT, a Modular Approach to Software Construction Operation and Test [1], became the MoD preferred design method for real-time software systems in 1978. MASCOT is not a programming language and therefore requires the support of a programming system. Moreover, MASCOT is independent of programming language which implies that within each language used to support a MASCOT application there must be techniques of realising MASCOT objects, operations and facilities. Languages possessing powerful features will enable the programmer to model certain of these MASCOT features more securely than less rich languages which may well present the programmer with an onerous task wherein there is much lost opportunity for compile-time checking.

CORAL66 [2] was adopted as the MoD standard computer programming language for real-time applications (the kind of system for which the use of MASCOT is encouraged) in 1970. It can thus be seen that CORAL66 predates MASCOT. MASCOT features can easily be mapped into CORAL66. It was realised, however, that an addition to the language would allow the features to be used straightforwardly whilst achieving improved clarity and program integrity. The success of the CORAL66 standardisation programme and resulting wide availability and use of CORAL66 suggested that an addition to CORAL66 was to be preferred to the design of a special MASCOT language.

AF-CORAL is an addition to CORAL66 which allows MASCOT ideas to be expressed conveniently, clearly and more securely than in CORAL66. The current version of AF-CORAL, referred to as AF-CORAL II [3], is based on the Official Handbook of MASCOT [4], which was issued by the MASCOT Suppliers' Association in early 1981. Annex A contains a summary of AF-CORAL in a tabular form similar to that used by the Official Handbook to summarise MASCOT. AF-CORAL II is controlled by a committee one of whose parent committees is IECCA (the MoD Inter-Establishment Committee on Computer Applications).

This paper assumes some familiarity with both CORAL66 and MASCOT.

2. MASCOT Features

MASCOT allows software to be partitioned into MASCOT Subsystems which are composed of Activities communicating through Intercommunication Data Areas (IDAs). Shared IDAs enable Activities from one Subsystem to communicate with Activities from another.

Subsystems may be controlled; thus, they may be STARTed to allow their constituent Activities to proceed, they may be temporarily HALTed and RESUMEd, or they may be TERMINATEd. Additionally, Subsystems are brought into existence by the MASCOT FORMing process, and are dismantled by DELETion.

Subsystems are constructed from MASCOT System Elements (SEs), which are CREATEd as needed and DESTROYed when redundant. System Elements are built from software templates and belong to one of two kinds: they are either built from an IDA template when they represent a passive data area used in communication (an IDA), or they are built from a Root Procedure, a template describing an active process which will eventually produce an Activity.

The software templates are known, unsurprisingly, as System Element Templates (SETs). They are coded in terms of programming language constructs and define the properties of a type of data area or active process. SETs need to exploit the primitives defined by MASCOT to enable the programmer to achieve MASCOT disciplines when controlling real-time interactions. MASCOT primitives also allow the programmer to monitor data and events, and to handle peripheral devices which are external to the software system. Templates, when they have been successfully compiled, are registered, and hence are made available for future use, by MASCOT ENROLment; they are removed from use by CANCELation.

The Official Handbook of MASCOT indicates that there are several categories of implementation and defines, for each category, the MASCOT constructs which must be present. There is, of course, a set of constructs present in all categories of MASCOT implementation.

Complete discussion of MASCOT is given in [4].

3. CORAL66 Features

CORAL66 is a block-structured, procedural language which provides the programmer with classical Algol60-like features: numeric types, arithmetic expressions, some control structure, parameterised procedures, etc. In addition, it provides independent compilation and statements which allow the programmer access to the computer hardware.

CORAL66 is fully defined in [2].

4. AF-CORAL Features

AF-CORAL provides language features, lacking in CORAL66, which allow the programmer to exploit MASCOT objects, operations and facilities more readily, thus encouraging the production of correct program text and providing greater opportunity for compile-time or construction-time checking.

While AF-CORAL provides a syntax and semantics for all MASCOT constructs and primitives it is not sensible to expect every construct and primitive to be implemented by a particular compiler (or pre-processor) if the underlying MASCOT implementation does not provide the necessary capabilities. AF-CORAL users should therefore be aware of the category of MASCOT implementation they are using.

The AF-CORAL features are, of course, designed to reflect MASCOT while being within the style and spirit of CORAL66.

AF-CORAL will evolve as MASCOT itself evolves ([4], Preface): the current version, AF-CORAL II, is defined in [3].

4.1 MASCOT Construction Database

Most programming languages, CORAL66 amongst them, regard a software system as a single monolithic program which is segmented at the programmer's peril since inter-segment checking is considerably reduced. MASCOT does not recognise the concept of the monolithic program but regards software as a set of essentially independent modules which interact according to given connectivity constraints. MASCOT would appear to demand not only that software should be highly segmented but also that there should be a good degree of checking between modules.

The Official Handbook of MASCOT postulates a database which holds all information necessary to the construction of a MASCOT application. This information is termed the Construction Database (CDB). An AF-CORAL compiler (or pre-processor) is concerned with gathering that information which will allow it to check the validity of AF-CORAL source text presented to it - such information is held in the CDB. Thus the CDB is, amongst other roles, the mechanism by which CORAL66 independent compilation is extended to a form of separate compilation.

4.2 Subsystems

AF-CORAL provides statements which allow the control of subsystems. The syntax of the statements conforms to that recommended in the Official Handbook of MASCOT. Subsystems are named, and their use can be policed by AF-CORAL compilers and pre-processors using information held in the CDB.

For example, we may write

```
START ( display );

HALT ( simulator input );
HALT ( simulator output );
```

to schedule the Activities in the Subsystem named "display", and to deny execution to the Activities of the Subsystems named "simulator input" and "simulator output" (at least until a corresponding RESUME is used).

RESUME and TERMINATE can be used similarly.

Before Subsystems are eligible to be STARTed they must be brought into existence by FORMing. Subsystems are built from System Elements, which have previously been created, and it is possible for AF-CORAL compilers and pre-processors to use the creation information, held in the CDB, to check that certain connectivity constraints are satisfied.

The syntax of FORM is that recommended by the Official Handbook.

For example

```
FORM simulator := radar simulator ( time,     constants,
                                    traffic,  display data ),
                  obey request    ( commands, traffic );
```

will bring the Subsystem "simulator" into existence by allowing the Activity "radar simulator" access to the data areas "time", "constants", "traffic" and "display data" and the Activity "obey request" access to the data areas "commands" and "traffic". Notice that the data area "traffic" is accessible to both Activities so that it provides communication between them.

MASCOT makes provision for the allocation of Activity priorities at FORM time. Should this method of priority allocation be implemented, AF-CORAL allows it to be exploited by extending the syntax of Subsystem declaration to include a machine dependent priority specification.

4.3 System Elements

System Elements too must be built. The syntax of the CREATion statement follows that recommended in the Official Handbook of MASCOT.

Examples of System Element creations are

```
CREATE (radar pool) constants;

CREATE (display chan) display data;
```

which create the IDAs "constants" and "display data" of type "radar pool" and "display chan" respectively.

In addition to these simple declarations AF-CORAL allows the programmer to supply values to satisfy size parameters which will therefore determine the actual size of the data area created. Initialising procedures may also be invoked.

For example

```
CREATE ( buffer [10] ) input buffer ;

CREATE ( buffer [20] ) output buffer ;
```

would cause two IDAs to be created, both fundamentally of the same structure but differing in size. An example creation which includes initialisation is

```
CREATE ( message channel ) transfer INITIALISE set up (0)
```

where an initialisation procedure "set up" is invoked with the value 0.

Furthermore, SEs may be supplied with the specification of machine dependent characteristics which are intended to allow such information as physical hardware mappings to be provided when the SE is created.

As has been seen, CREATion builds actual SEs which belong to a given type: types (with size and initialisation details as appropriate) are described by software templates.

4.4 Templates

A software template is the detailed specification of the properties of a MASCOT module in terms of programming language declarations and statements.

In AF-CORAL, the statements and declarations may be normal CORAL66 or they may be drawn from the set of AF-CORAL additions. The additions allow MASCOT primitives, operations and facilities, defined in the Official Handbook, to be written directly into the source text.

Templates are of two kinds: a template is either an IDA template, when it will describe the properties of a type of data area, or it is a Root Procedure, which is a template describing the properties of an active process which will eventually become an Activity.

4.4.1 IDA Templates

Within an IDA template a programmer may exploit MASCOT primitives to control real-time interaction. The MASCOT control queue (a data type in AF-CORAL) is available to the programmer and he can operate upon control queues using the control queue primitives. The control queue primitives have been imported directly from MASCOT but, of course, are written in CORAL66 style.

For example,

```
CONTROLQ reader q , writer q ;
```

declares two control queues which can then be used to control access to some resource using the mutual exclusion primitives so

```
JOIN ( reader q ) ;

LEAVE ( writer q ) ;
```

The cross-stimulation primitives are similarly available.

It is of course now possible to check at compile-time that objects of type CONTROLQ are used precisely as arguments to the MASCOT synchronisation primitives.

The usual CORAL66 data types are used to describe the data area of an IDA. CORAL66 data types may be parameterised to allow IDAs of the same structure but different sizes to be created.

The idea of a simple data area is not enough, however. MASCOT interaction implies that users of an IDA are denied direct knowledge of the structure of the area but instead are provided with a method of access to it. Such MASCOT Access Procedures may be written in AF-CORAL and are similar to CORAL66 procedures.

For example

```
INTEGER ACCESS PROCEDURE read;
BEGIN
  INTEGER a value;

  JOIN ( read q );
  WHILE empty DO
    WAIT ( read q );

  a value := data;
  empty := true;

  STIM ( write q );
  LEAVE ( read q );

  ANSWER a value
END access procedure read;
```

illustrates how the author of IDA source text provides a method to access the data contained in the IDA but at the same time protects that data from uncontrolled access.

AF-CORAL provides a method of decomposing the code of Access Procedures. Thus, a local procedure may be called from several Access Procedures within the same IDA specification. The names of Access Procedures (but not local procedures) are available to the authors of Root Procedures.

Special access to an IDA is provided by the Handler. It is intended that Handlers would be receiving data from a MASCOT Device and may therefore need to operate in a privileged mode. AF-CORAL provides a CORAL66-like syntax for this interrupt processing.

IDAs are characterised by the access mechanism interface they present to the Root Procedures that use them; indeed, the data area and the algorithmic details of the Access Procedures may be changed (provided the interface is not altered) with no consequent effect on any user Root Procedure.

4.4.2 Root Procedures

The Root Procedure is that SET which allows the programmer to describe the properties of a type of active process which eventually produces Activities.

Root Procedures interface to formal data areas, in an analogous fashion to the provision of formal parameters to CORAL66 procedures. The formal IDA parameters specify not only the type of data area to which the Root will have access, but further specifies the names of the Access Procedures available to the Root. Thus, direct knowledge of the structure of data areas is denied and access is in a controlled fashion by Access Procedure.

For example, the Root "copy" may be given access to an IDA in which data is being placed, and to an IDA into which it places processed data for consumption by another Activity:

```
ROOT copy ( CHANNEL (data) input  USING read;
            CHANNEL (data) output USING write );
```

The bodies of Roots may contain normal CORAL66 statements and declarations of local data. They may also contain constructs drawn from the AF-CORAL additions, which, of course, reflect MASCOT disciplines. Access Procedures are made available to a Root by inclusion in the Root USING list, they may then be used by coupling the Access Procedure name to the name of its formal IDA with the language word OF.

For example

```
ROOT copy ( CHANNEL (data) input USING read;
            CHANNEL (data) output USING write);
BEGIN
  INTEGER next;

  FOR ever DO
  BEGIN
    next := read OF input;
    SUSPEND;
    write OF output
  END read write loop

END root copy;
```

specifies an Activity which has the (rather simple) function of reading a datum for one data area and copying it to another, while forcing a re-schedule between these two operations.

Roots can, of course, be complex pieces of program which may require further procedural decomposition. AF-CORAL II desribes a method for decomposing a Root into Subroots.

4.4.3 Other MASCOT Primitives

It has been shown how the MASCOT control queue and its associated primitives may be utilised in IDAs and how the scheduling primitives are used in a Root. Other MASCOT primitives are applicable equally to IDAs and Roots. Such primitives may cause an Activity or Access Procedure to prematurely finish its processing slice and delay for a specified time, others allow the state of MASCOT monitoring to be affected.

5. Concluding Remarks

This paper has briefy described AF-CORAL II, an addition to CORAL66 which allows MASCOT to be exploited more safely and easily than in CORAL66 without additions.

It has been stated that not all implementations of MASCOT will offer the same facilities; implementations of AF-CORAL, which merely reflect the underlying MASCOT implementation, may, therefore, provide only a subset of the constructions described. Moreover, as the design method MASCOT is language independent the language provided by a MASCOT implementation may not be AF-CORAL at all; it may not even be CORAL66 based. In such cases the MASCOT programmer should expect to find advice in programming manuals which details precisely how MASCOT objects, operations and facilities are provided by that language system.

AF-CORAL II has been implemented in a commercially available product [5] which has been used for MoD contracts; a second implementation [6] is also proposed. An alternative approach is illustrated by the development of an AF-CORAL II pre-processor [7] which will allow MASCOT modules written in AF-CORAL II to be targetted to MASCOT700 [8] and to ASWE MASCOT [9].

RSRE are procuring a test capability [10] to allow them, on behalf of IECCA, to approve MASCOT Machine implementations for use on future MoD projects. The test software will be written using AF-CORAL II.

MASCOT and AF-CORAL II are contolled by JIMCOM - the Joint IECCA MUF Committee on MASCOT. The secretariat of JIMCOM and MUF (the MASCOT Users' Forum) are held by Computer Applications Division, RSRE.

Annex A

MASCOT Facility (referenced by Chapter of the Official Handbook)		AF-CORAL
Constructing (Chapter 3)	Building	CHANNELSPECs and POOLSPECs specify the properties of SETs using data items and various kinds of access mechanism. They are prepared by AF-CORAL compiler/linker or some form of preprocessor Access mechanisms are made available to Root Procedures by a USING clause and are written as a compound names with an OF. ENROL CREATE FORM
	Dismantling	CANCEL DESTROY DELETE
Controlling (Chapter 4)		START TERMINATE HALT RESUME
Scheduling (Chapter 5)	Synchronising	The data type CONTROLQ. JOIN WAIT LEAVE STIM
	Timing	DELAY TIMENOW
	Suspending & Terminating	SUSPEND ENDROOT

Device Handling (Chapter 6)		The data type CONTROLQ. CONNECT DISCONNECT STIMINT ENDHANDLER
Monitoring (Chapter 7)	Recording	RECORD
	Selecting	SELECT EXCLUDE

References

[1] Jackson K and Simpson H, MASCOT - A Modular Approach to Software Construction, Operation and Test, RRE Technical Note 778, October 1975.

[2] Woodward P, Wetherall P and Gorman B, Official Definition of CORAL66, HMSO, 1970.

[3] JIMCOM, Additional Features to integrate MASCOT with CORAL66, JIMCOM, July 1982.

[4] MASCOT Suppliers' Association, The Official Handbook of MASCOT, MSA, December 1980.

[5] System Designers Ltd., CONTEXT 3.1 (User Guide), January 1981.

[6] Study of MASCOT on the GEC 4000 Series Computers, GEC Computers Ltd, 1981.

[7] Stammers R, MASCOT Additional Features Preprocessor, SPL, 1981.

[8] Software Sciences Ltd/Ferranti CSL, MASCOT700 (User Guide), October 1980.

[9] Miles J, ASWE MASCOT Operating System (Reference Manual), ASWE, 1979.

DAT

ILM