CERN - Data Handling Division
DD/86/16
October 1986

USE OF VMS AND THE VAXSET FOR THE ADAMO DATA MANAGEMENT SYSTEM

P. Palazzi :CERN DD Division, Geneva, Switzerland
R. Brazioli :Heidelberg University, Heidelberg, W. Germany
S. M. Fisher :Rutherford Appleton Laboratory, Chilton, UK
W. R. Zhao :IHEP Academica Sinica, Beijing, China

Tel :+44-22-833897

Presented by P. Palazzi at the 1986 DECUS Symposium
Hamburg, September 1986

ABSTRACT

VMS and the VAXSet were used in developing ADAMO which is a software system written in FORTRAN-77 to define Entity-Relationship[1] (E-R) data structures, map them onto tables and manipulate them from FORTRAN programs. The name ADAMO is an acronym for Aleph DAta MOdel, where Aleph is an experiment in preparation at CERN, the European Laboratory for Particle Physics.

The main aim of this paper is to show how we have made use of VMS and the VAXSet to build ADAMO, and to build the tools to allow it to be ported to other systems.

THE ADAMO SYSTEM

ADAMO is a software system to define Entity-Relationship (E-R) data structures, map them onto tables and manipulate them from FORTRAN programs. It is especially suitable for handling numeric data within algorithms. Tables can be transferred to and from a relational data base. The Structured Analysis method[2] has been extended to allow flows appearing on data flow diagrams to have E-R data structures as components.

The TAP (TAble Package), is a package of FORTRAN callable subroutines to manipulate E-R and/or tabular data. It is a fundamental part of the system, as not only does application code use the TAP, but so do all the modules of the system. During execution, active tables reside in memory for fast access, and can be stored and retrieved from Direct Access Files.

The system was built because of the growing complexity in the data processing requirements of particle physics experiments. The experiments have more data and the programs require larger and more complicated data structures. More people are involved in designing, and writing and using the programs. The experiments have a longer lifetime, so the software to go with them must be able to stand the test of time. Keeping the program documentation up to date is also increasingly difficult.

There were of course certain constraints. Algorithmic applications are especially important in experimental particle physics. High execution speed of the final code is very important. Portability of the system kernel and of the application code is vital, and we wished to have the full system available with a good interface on VAX/VMS and IBM VM/CMS which have been chosen as the development systems for the Aleph experiment. We had decided to continue with FORTRAN-77 which is the traditional language in our field, and at the moment there seems to be no practical alternative.

We felt it to be very important to distinguish between the source and target software system. For those developing the ADAMO system the source was VAX/VMS and the targets VAX/VMS and VM/CMS except for the TAP itself for which the target is any machine with a FORTRAN-77 compiler. ADAMO users take our target as their source and produce code to run on any machine where the TAP will run i.e. a machine with a FORTRAN compiler.

To base our data manipulation language, the TAP, upon FORTRAN could have been done in three ways: modify a compiler, write a pre-compiler or define a subroutine package interface. Compiler modification would have been a major effort, and would have been non-portable. A pre-compiler makes application code development much harder because the output of the pre-compiler as seen by the debugger is not readily recognisable by the user. So the third option of a subroutine package was adopted, though a pre-compiler may come later for established application code where execution speed is the overriding consideration.

Main Components of ADAMO

The system consists of independent processes, each performing a specific task, which can be activated individually, and communicate through standard interfaces. Figure 1 shows the overall architecture of the system. These data flow diagrams have been prepared and checked by our own tools. We summarise below the main processes. Those processes which were developed to assist mainly in porting the system are described in the next section.

CSP The Conceptual Schema Processor reads the Data Definition Language and creates or updates the corresponding Data Dictionary.

LID lists in tabular form the contents of a Data Dictionary.

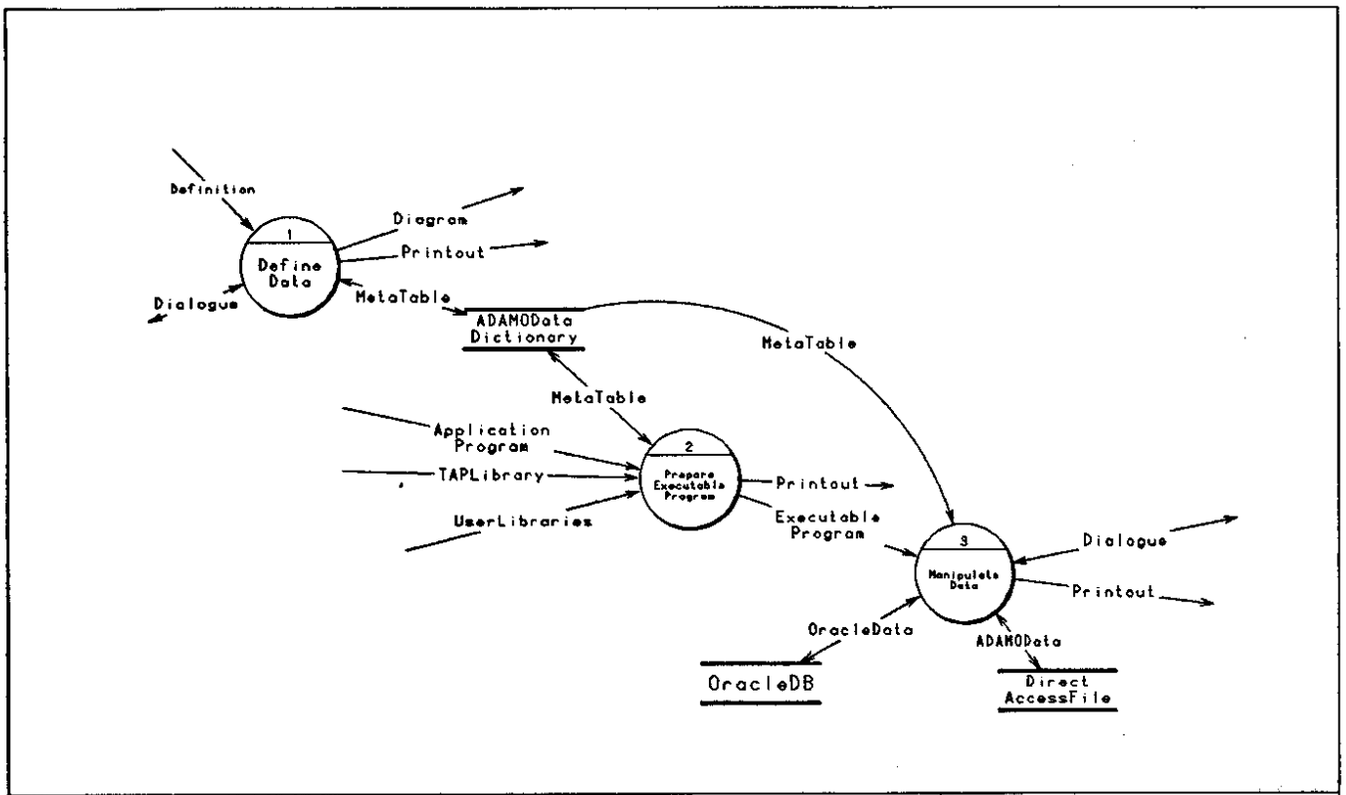
FLB the Fortran Language Binder produces FORTRAN common blocks and an initialization routine to be inserted in an application program using the TAP.

TIN provides interactive access to TAP primitives. Each TAP subroutine corresponds to an interactive command. On-line help is provided.

TOR converts ADAMO tables to/from the ORACLE RDBMS.

DFA the Data Flow Analyser checks the consistency of a family of Data Flow Diagrams and their compatibility with given E-R data structures. It can take as input the Data Flow Diagrams entered through the Tektronix SA Tool[3].

DFP draws data flow diagrams. Inconsistencies detected by the DFA can be indicated by icons on the diagrams themselves.



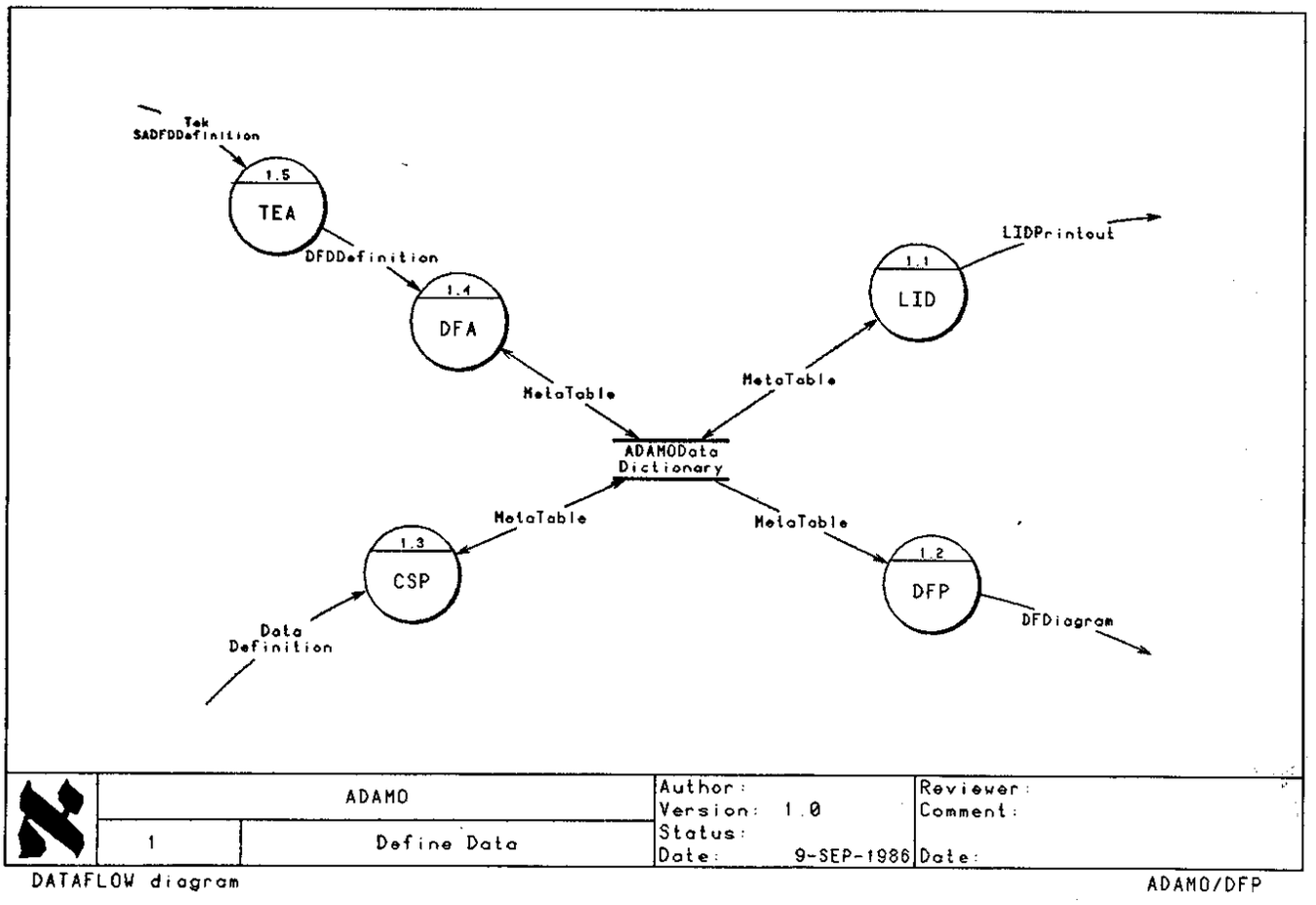
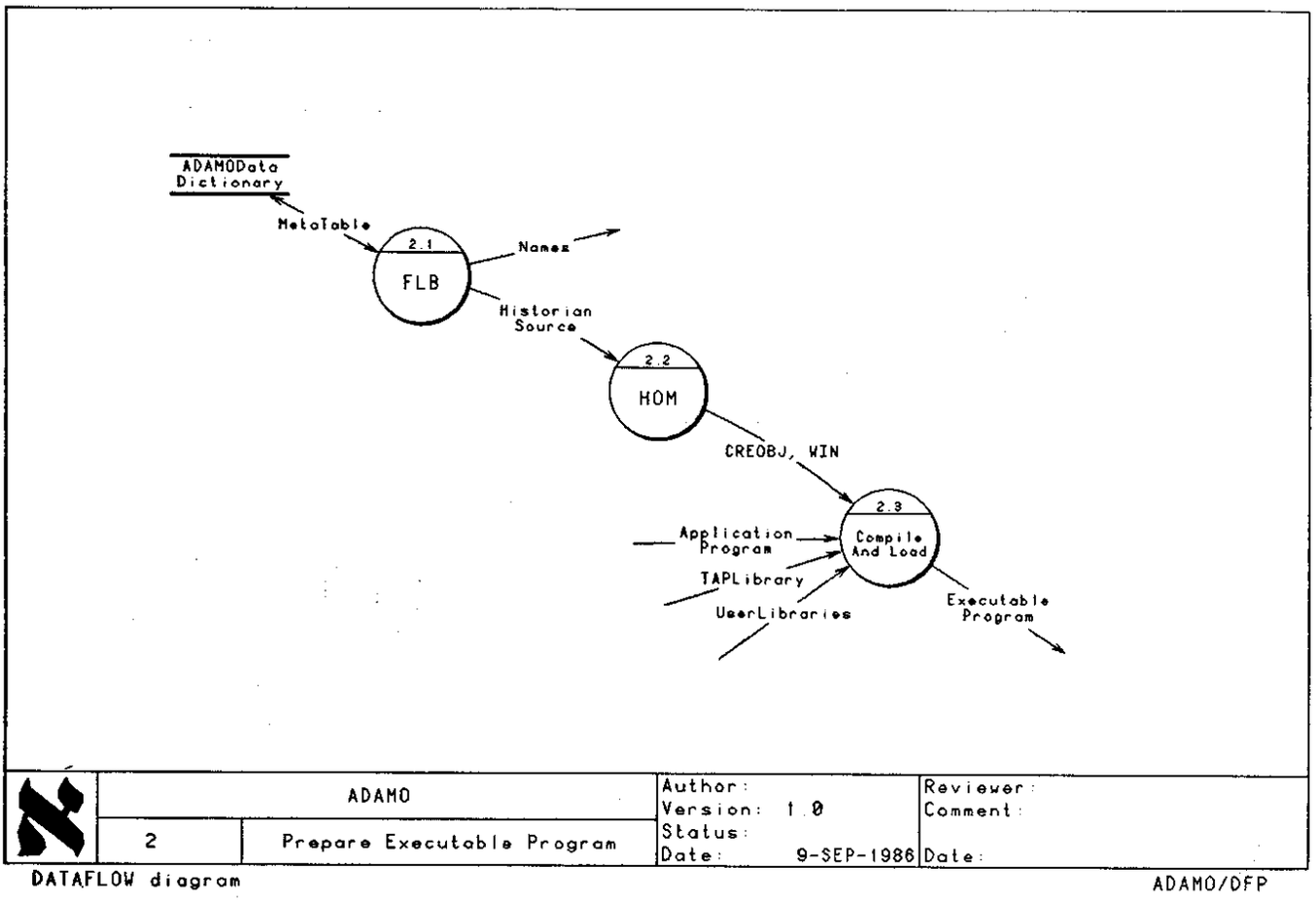


Figure 1a and b: Data Flow diagrams of the ADAMO system



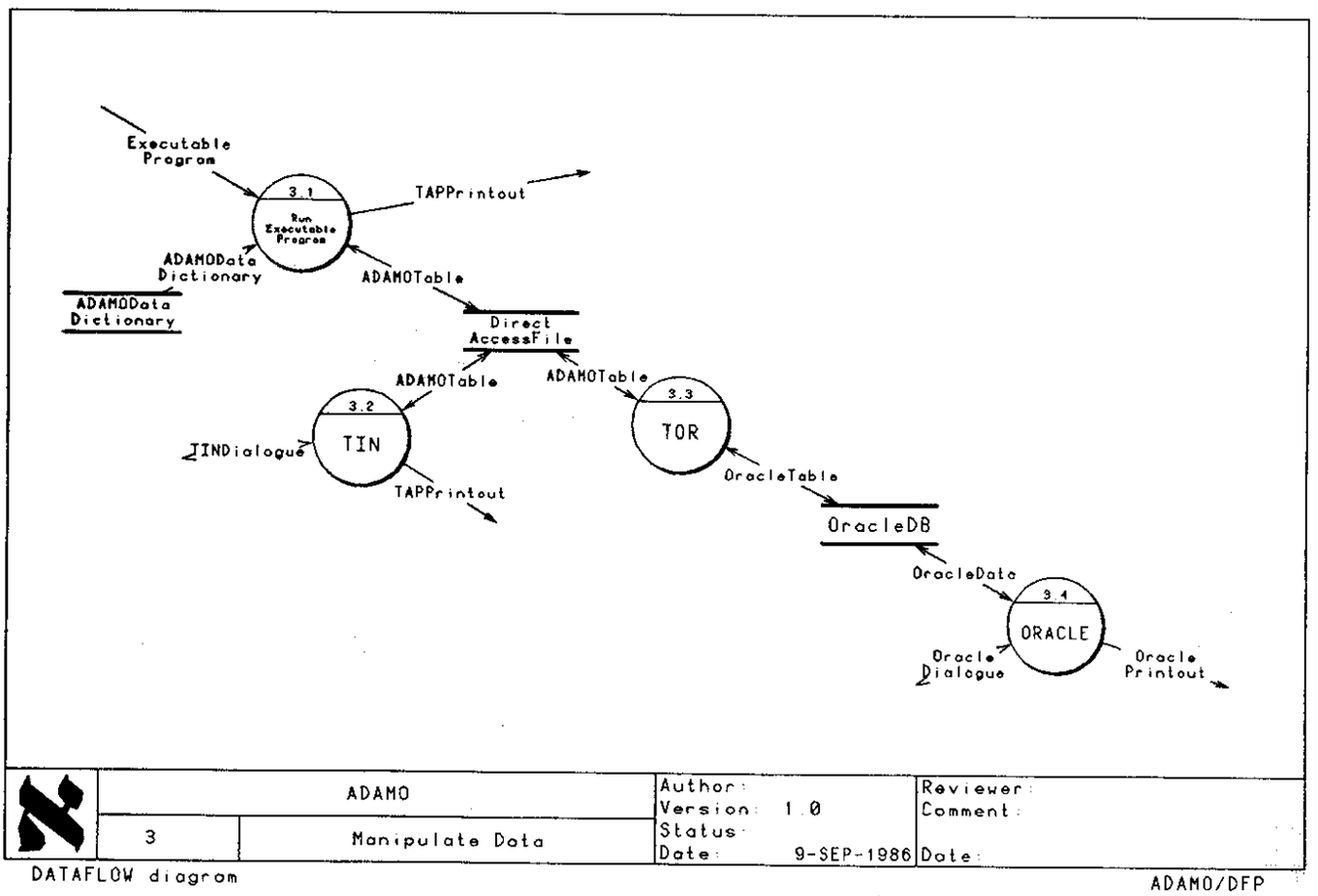


Figure 1c and d: Data Flow diagrams of the ADAMO system

Utility Components

These tools are not related to the data dictionary but are code management tools, useful for developing software which is targeted for a different computer system.

MOH converts a group of FORTRAN source files with INCLUDE statements to a machine independent representation compatible with the source format of the HISTORIAN[4] code management system and held in a single file. The input to the program is shown in figures 2 and 3 and the output in figure 4.

```
      SUBROUTINE EXAMPLE
      IMPLICIT NONE
C
      INCLUDE 'Vehicle.INC'
      INCLUDE 'ADAMO$USR:PARTAP.INC'
C
      Vehicle_ID = NEXT       ! NEXT Vehicle
      CALL INSENT(Vehicle)    ! Create new entity Vehicle
      END
```

Figure 2: VAX FORTRAN file

```
      COMMON /Vehicle/          Vehicle
     +                         ,Vehicle_ID
     +                         ,Vehicle_TAIL
C
      INTEGER                   Vehicle
      INTEGER                   Vehicle_ID
      INTEGER                   Vehicle_TAIL
```

Figure 3: VAX FORTRAN INCLUDE file

HOM converts the output of MOH into individual files and INCLUDE files. The program also exists on IBM VM/CMS where it is written in REXX and produces the corresponding INCLUDE files for the IBM VS FORTRAN compiler.

FOS converts between different dialects of FORTRAN. It is able to convert back and forth between VAX and IBM format provided that only the following extension are used: in-line comments, IMPLICIT NONE and long variable names provided that they are known to the data dictionary and are of the form EntitySet_Attribute.

```
*DK FOREG
      SUBROUTINE EXAMPLE
      IMPLICIT NONE
C
*CA VEHICLE
*CA PARTAP
C
      Vehicle_ID = NEXT       ! NEXT Vehicle
      CALL INSENT(Vehicle)    ! Create new entity Vehicle
      END
*CD VEHICLE
      COMMON /Vehicle/          Vehicle
     +                         ,Vehicle_ID
     +                         ,Vehicle_TAIL
C
      INTEGER                   Vehicle
      INTEGER                   Vehicle_ID
      INTEGER                   Vehicle_TAIL
*CD PARTAP
C ..............................................................
C
C        system-defined parameters for the Table Package Programmer
C
.
.
```

Figure 4: VAX FORTRAN in HISTORIAN format

The input may be the output of the MOH program. Figure 5 shows the result of processing the file in figure 4.

SAC selects appropriate code from a file. It is useful to be able to keep in a single source file the code for different target machines, and program variants. An example of such a variant for the TAP is input parameter checking in user callable subroutines. As FORTRAN, unlike some other languages, has no conditional code facilities, the SAC was written to Select Appropriate Code from a source file. Figures 6 and 7 show an extract from a file with some conditional code. Note that as the file produced by the SAC has no less information than the input file, the tool is able to convert the code in figure 6 to that of 7 and vice versa. The syntax chosen is compatible with HISTORIAN.

USE OF VMS AND VAXSET TO DEVELOP ADAMO

We decided to develop ADAMO using the facilities of VAX/VMS and to take advantage of the best tools we could get and not restrict ourselves to tools which are portable where portability is not relevant.

VAXset in ADAMO Development

There are some overheads associated with all tools, as one must take the time to learn to use them effectively. This time was, we feel, well spent and extensive use was made of most of the tools.

```
*DK FOREG
      SUBROUTINE EXAMPLE
      IMPLICIT LOGICAL(A-Z)
C
*CA VEHICLE
*CA PARTAP
C
      VHCLID = NEXT
C|                      NEXT Vehicle
      CALL INSENT(VHCL$$)
C|                             Create new entity Vehicle
      END
*CD VEHICLE
      COMMON /VHCL$$/            VHCL$$
     +                           ,VHCLID
     +                           ,VHCLTL
C
      INTEGER                    VHCL$$
      INTEGER                    VHCLID
      INTEGER                    VHCLTL
*CD PARTAP
C ..................................................................
C
C        system-defined parameters for the Table Package Programmer
C
.
.
```

Figure 5: IBM FORTRAN in HISTORIAN format from figure 4

NOTES is a computer conference system which was used by the development team. It was found useful even though they all live in the same room.

Because thoughts are typed in on a terminal rather than being explained verbally, one has more chance of saying something which is concise, coherent and well thought out. This saves everyones time and provides a written record.

It might have resulted in a different style of working, however its impact was reduced because it can be a very slow way of making progress, one is restricted by not being able to present diagrams, and there is no mechanism for maintaining lists of notes. Examples of lists of notes might be those which one wants to respond to later, or a those which contain good ideas but should not be considered for 6 months.

A minor criticism is that the tool takes too long to start up, and it should be easy to request the following "Update these classes and if nothing new return to DCL".

LSE the Language sensitive editor is a major advance on EDT, though it retains the EDT keypad so is very easy to learn. The main advantage during development of ADAMO was the COMPILE/REVIEW facility which proved very useful in quickly identifying and fixing compilation errors. Other useful features are the two windows and multiple buffers.

```
.
.
      LOGICAL FLAG,FOKIND
C
*IF DEF,PRECOND
C
C-----------------   PARAMETER VALIDITY CHECKING  -------------
C
      IF (FEXTAB(TAB(1))) GOTO 888
      IF (FOPACC(TAB(1),ACC)) GOTO 888
      IF (OUTRAA(TAB(1),ACC,C,C)) GOTO 888
C
C check if the index is updated
C
C     IF (FOKIND(TAB(1),ACC)) GOTO 888
C
*EI
C
C-----------------------   ACTION  ------------------------------
C
      RIDBS=IZ(TAB(1)-RIDPOS)+RIDBAS
.
.
```

Figure 6: extract from a file with the PRECOND flag on

```
.
.
      LOGICAL FLAG,FOKIND
C
*IF,DEF PRECOND
**DEL** C
**DEL** C-----------------   PARAMETER VALIDITY CHECKING  -------------
**DEL** C
**DEL**       IF (FEXTAB(TAB(1))) GOTO 888
**DEL**       IF (FOPACC(TAB(1),ACC)) GOTO 888
**DEL**       IF (OUTRAA(TAB(1),ACC,C,C)) GOTO 888
**DEL** C
**DEL** C check if the index is updated
**DEL** C
**DEL** C      IF (FOKIND(TAB(1),ACC)) GOTO 888
**DEL** C
*EI
C
C-----------------------   ACTION  ------------------------------
C
      RIDBS=IZ(TAB(1)-RIDPOS)+RIDBAS
.
.
```

Figure 7: same file as in figure 6 with PRECOND flag off

A minor criticism is again that it takes too long to start up. It would also be nice to be able to request placeholders and tokens which excluded the VAX extensions to FORTRAN. LSE lacks global change features. Substitution should be much more powerful, it should be possible to make it case sensitive, column sensitive and to accept multiple wild characters. In addition it would be convenient to be able to use tabs while editing a file but to produce a file without tab characters. This is possible by specifying the language to be FORTRAN but then properties are associated with column 6 which may be inappropriate.

CMS is a powerful code management system. All the ADAMO source material is maintained with this tool. It is very convenient for keeping a record of what has been done, and by whom. For each release of the ADAMO system we are readily able to extract the source for one component. This we do in general by defining a group to correspond to the current composition of a tool then inserting that group into a class before moving on to the next version.

However it has a few annoying features; the main one being the prompt for remarks. When something is being created or an element replaced then the remark is valuable as it is permanently associated with the object and null remarks should perhaps not be allowed. In other cases however, when remarks are just stored in the history file no prompt should be produced. It would also be convenient if "replace" did not produce a new generation if there are no differences. Currently it produces a message - but by then it is too late.

The other more general problem with having a code management system of this sort is that to avoid making costly mistakes one must be careful not to have many files fetched from CMS. This is in conflict with the need for the debugger, PCA and the SEARCH command to access source material. A solution might be to build CMS more deeply into VMS.

SCAN is a language which is able to recognise patterns in text and produce a modified output. It was used in writing the SAC, which was described earlier, and is a tool to handle conditional code. The language was considered to be optimal for this type of application.

PCA the performance and coverage analyser, was found to be extremely useful. The two basic uses, performance - making the program faster, and coverage - ensuring that a test exercises the code, are quite distinct and in some respects do not fit well within the one tool. It works well for performance but is less convenient for coverage.

We used it extensively for identifying the critical parts of our TAP package and found that with a little practice it was very easy to identify the parts of the code to optimise.

It would be nice for a certain routine to determine what fraction of the time it is called by which routines. Working with shareable images is clumsy.

We made use of the coverage facilities as part of our testing of the tools. It did allow us to do the job, but as it is not possible to ask which routines have one or more paths not covered, it is very difficult to know if coverage of the program by a test has become better, worse, or stayed the same.

DTM the Dec/Test Manager organises the tedious job of testing. It is very convenient and because of this does encourage better testing. A range of tests have been set up for our tools and for the Table Package. We are confident that this will greatly inhibit the creation of new bugs and allow us to deliver a better system.

MMS the module management system has been used, but there was some delay in getting started with it because it needs a tool to generate the dependency files. This we have written for dependencies of a .EXE file upon FORTRAN and DDL files.

VMS AND VAXSET AS ONE OF THE TARGET ENVIRONMENTS FOR ADAMO.

We have already described the use made of VMS and VAXset in developing the system, now we explain how they are being used in the target environment.

VMS in the Target Environment

HELP: all documentation was stored as small source files which were assembled and formatted as paper documents or as HELP files. Formatting was performed by a text processing system with a small set of specially written macros, and a post processor. The standard format of VMS help was found to be inadequate as there was not enough information at the top level. Our help files (see figure 8) were constructed such that all parameters and qualifiers got a one line summary on the top level.

```
CSP

   Process DDL files to update or produce a data dictionary ADD

   CSP DDLFile[,...]              : input DDL files
       [ADDFile]                  : data dictionary file
         /[NO]CREATE              : create a new data dictionary <NOCREATE>
         /[NO]LIST[=filename]     : there is a listing file <NOLIST>
         /[NO]LOG                 : log session on terminal <NOLOG>
         /[NO]ODD                 : write to ODD <ODD>
         /[NO]CHECK               : perform validity check on DD <CHECK>
         /[NO]SUPERSEDE           : supersede data definitions <NOSUPER>
         /[NO]INCOMPLETE_DD       : output DD though incomplete <NOINC>
         /ESET_LENGTH=int         : maximum length of Eset name <16>
         /ATTRIBUTE_LENGTH=int    : maximum length of Attribute name <16>


   Additional information available:


Press RETURN to continue ...
```

Figure 8: An ADAMO VMS/HELP file

The way in which HELP is formatted on the screen by VMS is very wasteful. This should be improved upon. It would also be nice to be free from the tyranny of the hierarchical HELP structure.

CLI: each tool had its own command to invoke it and CLI was used extensively. For each tool just one VAX/VMS specific subroutine was written. This made it simple to write the necessary part for each tool to interface to other operating systems, specifically VM/CMS

VAXset in the Target Environment

PCA is being used to identify CPU intensive areas of user code, where a standard sequence of steps may be performed to obtain the desired performance. Eventually replacing subroutine calls to the TAP by hand coded access to the tables, generally through statement functions.

LSE proved valuable for teaching ADAMO users the syntax and semantics of the ADAMO DDL. For this we defined a language in the LSE sense but were hindered by not being able to create a diagnostic file as its structure is not published. We also wrote placeholders and tokens to assist with writing our DML, that is the FORTRAN with calls to the TAP package. An example of one of our tokens is shown in figure 9.

```
DEFINE TOKEN SELECTION -
     /LANGUAGE=FORTRAN -
     /DESCRIPTION="Select rows by an index" -
     /TOPIC=""

     "CALL SELTAB({tab},{ind},{cur1},{cur2})"
     "DO {lab} {cur}={cur1},{cur2}"
     "     CALL FETTAB({tab},{ind},{cur},{ok})"
     "     {executable_statement}..."
     "{lab}     CONTINUE"

END DEFINE
```

Figure 9: LSE as an aid in writing TAP code

NOTES: a conference has been created to allow ADAMO users to exchange ideas.

CONCLUSIONS

While developing the ADAMO system on VAX/VMS, we were able to make use of several VAXset tools as they became available. They were found very useful for software development and we now regard them as part of our environment and thus rely upon them also for the maintainance phase of our system.

The loosely coupled architecture of ADAMO makes it fairly easy to integrate its components in a work-station environment. We hope to be in a position to do so soon, providing good interactive graphical interfaces to our design tools.

ACKNOWLEDGEMENTS

We are grateful to the Digital Equipment Corporation and Alan Silverman from CERN for making the components of the VAXset available to us at an early date.

We also wish to thank John Harvey and Gottfried Kellner for reading and commenting upon this paper.

REFERENCES

1. P. Chen, The Entity-Relationship model - Toward a Unified View of Data, in ACM Transactions on Database Systems, Vol. 1, March 1976, Pages 9-39.

2. T. DeMarco, Structured Analysis and System Specification, Yourdon Press, 1978.

3. Tektronix Structured Analysis Tools Users Manual, Tektronix inc.

4. HISTORIAN PLUS is a source code management system, available from OpCode Inc, P. O. Box 10998-537, Austin Texas 78766-1998