DESY DV-78/03
May 1978

Improvement of Programming Efficiency[+] in Medical Image Processing by a Dialog Language

G. Pfeiffer and K.H. Höhne
Deutsches Elektronen-Synchrotron DESY

Abstract

A system for image processing in medicine should on the one hand provide an easy and safe application of proven methods of picture analysis. On the other hand a smooth path for improvements or new applications should be provided. It is argued that the required system and programming flexibility can be achieved by a dialog language, which in addition provides for a better transparency and documentation of the resulting software.

The system described which has been implemented on a PDP11-Computer consists of two languages, a high level problem oriented dialog language and a low level machine oriented programming language. The paper summarizes the high-level dialog language. Particular features include specific interactive facilities, versatile display control and menu techniques, a powerful procedure concept and the use of default values and current representations of data types in procedures.

Zusammenfassung

Ein System zur Verarbeitung medizinischer Bilder sollte die Möglichkeit bieten, erprobte und ausgetestete Verarbeitungsmethoden einfach und sicher anzuwenden. Gleichzeitig sollten gut handhabbare Möglichkeiten bestehen, Verbesserungen oder neue Methoden im System zu implementieren.
Die Verwendung einer Dialogsprache ist ein geeignetes Hilfsmittel, die angestrebte System-Flexibilität zu erzielen, und darüber hinaus kann sie eine bessere Transparenz und Dokumentation der erzeugten Software gewährleisten.
Das hier beschriebene, auf einem PDP11-Rechner implementierte, System besteht aus zwei Sprachen, einer höheren problemorientierten Dialogsprache und einer niederen maschinen-orientierten Programmiersprache. Es wird eine Übersicht über die wesentlichen Eigenschaften der Dialogsprache gegeben. Diese umfassen spezielle interaktive Fähigkeiten, 'Menu'-Techniken, problem-orientierte Datentypen, aktuelle Darstellungen von Datentypen und ein vielseitiges Prozedurkonzept, insbesondere mit der Möglichkeit, Parametern 'Default'-Werte zuzuweisen.

[+] Accepted for presentation at the First Congress of the European Federation for Medical Informatics, Cambridge 4 - 8 September 1978.
Research supported by grant DVM 130 of the Bundesministerium für Forschung und Technologie.

1. Introduction

Image processing is of growing importance among the vaiety of computer applications in medicine. Many applications have been published, mostly in Nuclear Medicine, Radiology or Cytology. In our laboratory we are presently developing a system for the analysis of X-ray angiograms[1]. The overall system is subdivided into three major parts, the

- system hardware,
- system software and the
- application software.

In this paper we will concentrate on our approach to an appropriate system software. A detailed hardware description can be found in ref. 2. An overall view of the hardware may be obtained from the block diagram in Fig. 1. An X-ray picture series taken from a standard TV-interface (50 frames/s) is digitized in real time (256 x 256 pixels/frame) under control of a host computer. The digitized images are transferred to a mass storage device for further investigation. Image presentation is done on a colour TV monitor with light pen, track ball and key board. Results of first applications are described in ref. 3.

The *system software* represents an appropriate abstraction level to hide special hardware pecularities from the user as well as to provide application oriented tools for interaction with the hardware components. From experience with a system we had previously developed which included the interactive analysis of scintigraphic images[4,5] we learned that the complexity of the man-machine interaction is a major problem in the design of a system for large scale application of image processing for use by physicians in a hospital. One has to face a double task:

- *Development* of algorithms for new applications. Typical problems are varying modes of data acquisition, filtering algorithms or methods of data presentation.

- *Integration* of proven algorithms into a system which must not only be operational in daily routine, but sufficiently flexible to be applied to new areas of scientific research. The system is constantly subjected to modifications. These may be changes to already available programs (e.g. modifying a smoothing algorithm) or even entirely new applications to be fit into the system.

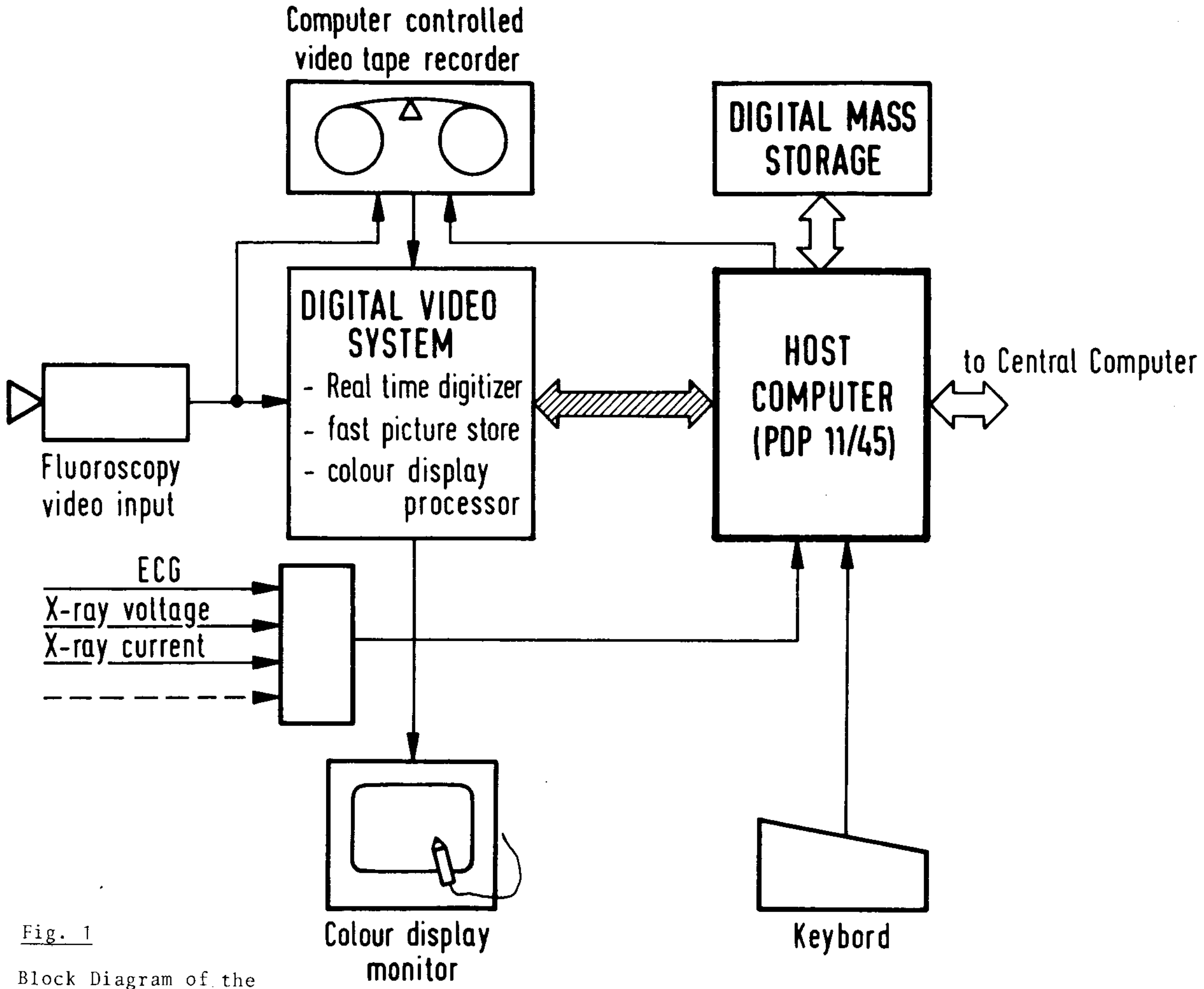

Fig. 1

Block Diagram of the Image Processing System

As has been pointed out by Kupka[6], the underlying issue is that of *procedural* and *non procedural* problem solving. The former involves transformation of a special problem to a mathematical algorithm, which in turn is represented by a program in an appropriate programming language. Non procedural steps, on the contrary, are concerned with developing, testing or modifying algorithms and they imply last but not least system maintenance tasks such as adapting the system to changing demands.

In this respect we felt that a problem oriented, conversational high level language could improve programming and system flexibility in image processing.

Available dialog languages, which we investigated, proved to be deficient for our application in one or more of the following respects:

- Low efficiency due to interpretative execution.

- Lack of specific facilities for image processing (problem orientation).

- Restricted portability due to specialized hardware and to operating system dependencies.

In particular MUMPS, which is widespread in medical data processing, was not considered suitable, mainly due to its lack of execution speed and problem orientation for image processing.

In this paper we will describe a dialog language which has been developed from experience with an earlier version[7]. Its purpose is to contribute to the solution of at least the first two of the difficulties mentioned above.

2. Requirements to the Language

Some design conditions are common to other conversational systems. A detailed discussion can be found in refs. 6 and 8. Further general conditions, which are specific to the image processing application, have to be considered:

- Efficiency - Response times must be short enough not to frustrate the user. This forbids interpretative execution of operations on large amounts of data (e.g. transformations of image matrices).

- Simplicity - Intricate concepts should be avoided in favour of simple and concise rules, so that untrained but interested physicians or technicians can use the language.

- Wide range of problems - A high level language may be inapplicable to some problems, e.g. acquisition of data from fast processes. Therefore, means must be provided to include applications outside the primary scope of the language.

- Wide range of users - A strict separation of routine and scientific use is in general not possible. The system has to satisfy users which only want to push buttons and users writing sophisticated application programs.

- Specific hardware support - The instruction sets of special processors with special memories must be accessible from the high level language.

3. System Concept

Fig. 2 shows the general structure of the system. Essentially it covers two different languages embedded in a common frame:

- a *low level* machine oriented programming language[9]

and - a *high level* problem oriented dialog language.

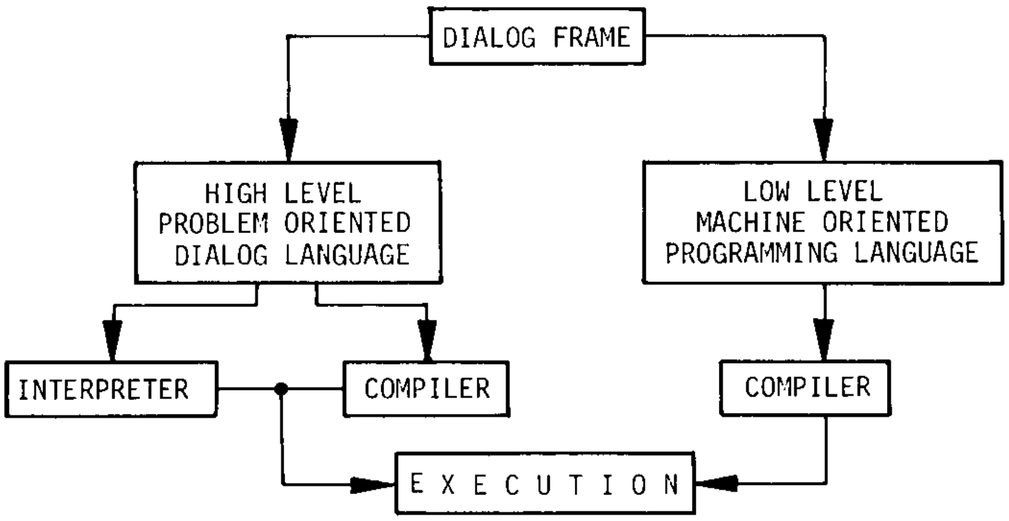
DIALOG FRAME
HIGH LEVEL PROBLEM ORIENTED DIALOG LANGUAGE
LOW LEVEL MACHINE ORIENTED PROGRAMMING LANGUAGE
INTERPRETER
COMPILER
COMPILER
E X E C U T I O N

Fig. 2 Structure of the Dialog System

The dialog frame consists of operations and data structures common to both languages.

The low-level language belongs to the class of PL360-like languages[10] and combines the style of a high-level language with the efficiency of an assembler language. It will be used when the application problem requires especially efficient code or explicit access to system hardware. In the dialog system the low-level language serves a double purpose:

- The system is *implemented* in the low-level language. Thus system modifications, which neven can be totally avoided, are achieved more easily.

- Procedures of the low-level language may be *linked* to the high-level language during a dialog session. Thus in applications not suited to high-level programming one can switch to low-level programming. However, use of the low-level language still requires considerable knowledge of computer hardware. In practice, therefore, its use will certainly be restricted to experts.

Although the low-level language and the high-level language fulfill entirely different requirements, they have a set of common language constructs. In fact there exists a subset of syntactic rules which have exactly the same definitions in both languages. Essentially these are the rules for

- formulation of expressions and

- program control structures.

The common subset is limited on either side by the machine orientation as well as the problem orientation of the respective language. Nevertheless the similarity of the two languages can facilitate the rewriting of tested algorithms from high to low level language.

Details of the low-level language are described in ref. 9. In what follows we concentrate on the *high-level* language, since it is used in most applications. To enhance efficiency, besides the interpreter a compiler will be provided. The user decides the mode of translation from the actual problem structure. While the interpreter offers the full scope of language definitions, usage of the compiler will impose some restrictions on the user, e.g. the dialog features are obviously not available.

4. Elements of the High-Level Language

The most important language features are summarized below. More details are given in ref. 7. Some concepts used in this language have been taken from other languages[8].

Declaration and Data Types

For numerical computing and string processing the data types BOOL, STRING, INTEGER and REAL are available. Since our application is image processing, we provided the problem oriented data types IMAGE and FRAME. A series of images of e.g.an angiographic study is represented by a three-dimensional image in the language. Frames represent rectangular or arbitrarily shaped areas of an image (e.g. a particular vessel in a kidney might be an arbitratily shaped frame). The frames may have one, two or three dimensions.

By indexing images with frames one has access to sub-images. For example, from a series of 200 images (256 x 256 pixels) a sub-series with a frame given by the parameters x = 10-200, y = 1-256, z = 20-100 could be extracted like

```
IMAGE[10-200,1-256,20-100]             or by using key words for axis
                                       orientation
IMAGE[%Z=20-100, %X=10-200, %Y=1-256]  or just
IMAGE[FRAME]                           if FRAME represents the desired
                                       three-dimensional region.
```

Declaration of objects may be done implicitly from the context or explicitly by a *DECLARE* statement, e.g.:

```
DECL I1,I2:IMAGE, X,Y,Z:REAL, TABLE[100]:INTEGER
```

User Defined Data Types

Users may define their own problem oriented data types, a feature well known and appreciated from other programming languages. For example, the user could combine a picture taken from a patient with his identification into a new structured data type, called RECORD:

```
STRUCT RECORD=IDENT:PERSON, ITEM:IMAGE
STRUCT PERSON=NAME[10]:STRING, AGE[3]:INTEGER
```

RECORD itself is composed of another structured type, called PERSON and an object of primitive type IMAGE. The components of the structured types can be accessed by numerical indices or by identifying names, as in the example IDENT, ITEM, NAME, AGE.

Current_Variables

When working intensively with one object, e.g. trying a number of manipulations on an image, it is awkward to refer to this object always by its name. Better practice is to identify the object *currently* in use by a special symbol. Thus there may exist a current representation of each data type, primitive or structured. It is identified by the type name and the character $, e.g.

```
IMAGE$[FRAME$]+REAL$
```

means addition of the current real number to an area of the current image, defined by the current frame.

Reserved_names

Some special purpose memories, e.g. two fast 256 x 256 8-bit memories for images are part of the hardware. In the dialog language they are represented as pre-defined objects with reserved names. To avoid conflicts with user-defined names the special sign '%' always must precede a reserved name. The hardware supplied image memories for example are integrated into the language as

```
%MEM1[256,256,1]and %MEM2[256,156,1]
```

of type IMAGE.

Control_Structures

A well-proven set of control structures was selected for the system. Iteration is provided through a FOR, WHILE or REPEAT loop, conditional execution through an IF...THEN...ELSE, whereby the ELSE clause is optional.

Examples:

```
      FOR 10 → I UPTO 100→ UP DO<STATEMENT>
      IF IMAGE% → I<LEVEL THEN 0 → I ELSE IxI → I
```

The last example requires comment. An image is assigned to I. The condition will be tested for each element of image I and the THEN or ELSE exit is taken depending on the result of the test. Thus execution of the condition for an image involves implicit execution of a loop. The above statement squares each image element above a given threshold and zeroes the rest.

Case selection is performed by a sequence of IF...THEN...Statements, guarded by DOCASE...ODCASE, e.g.

```
      DOCASE
         IF   RESULT EQ A THEN PRINT A
         IF   RESULT EQ B THEN PRINT B
         ELSE PRINT 'FALSE'
      ODCASE
```

The first TRUE condition is selected for execution.

A simplified loop structure allows iteration in an array without specifying any loop indices, e.g.

```
      DECL ARRY[256]:INTEGER
      LOOP ARRY[X] DO X → ARRY
```

Iteration is performed for all elements of the object ARRY. The dummy name 'X' is only used to identify the current index in each iteration step. This loop stores the numbers 1 to 256 in ascending order into ARRY.

Expressions

Expressions are evaluated strictly from left to right, without hierarchy of operators. Parentheses may be used to override this rule. Assignment also obeys these rules, as the example indicates:

```
      XY → LEVL + (FPARM+100. → X) → XY
```

The result is LEVEL=XY and X=FPARM+100. and XY=LEVEL+X.

Procedures

Procedures are divided into two classes,

- Internal Procedures (PROC) and

- External Procedures (XPROC).

Internal procedures (referred to as procedure) are program modules generated interactively in the high-level language. *External procedures* form the link between dialog programs and low-level programs. Internal and external procedures are declared in a parallel manner. The handling of parameters is the same for both cases.

Depending on the context the user may select from several notations for the passing of parameters:

- *functional* notation

-- *operational* notation, for up to two parameters

- *keyword* notation, for selection of specific parameters

This is illustrated by the following procedure:

```
PROC SUB (FIRST,SECOND:IMAGE)
IF ABS(FIRST-SECOND) LT LEVEL THEN 0 → FIRST
RETURN
```

If applied to images belonging to an angiographic series this procedure would eliminate static regions, i.e. only dynamic changes above a given intensity level would be preserved.

The procedure may be called in one of the following equivalent ways:

```
SUB(I1,I2)
I1 SUB I2
SUB with FIRST=I1 & & SECOND=I2
```

Procedure SUB can be transformed to an external procedure by the declaration:

```
XPROC SUB(FIRST,SECOND:IMAGE): XSUB
```

where XSUB is an arbitrary name referring to an entry point in the low-level language.

Thus depending on the efficiency required, one may choose to generate either a procedure in the high-level language or an external procedure in the low-level language. External procedures turned out to benefit not only the user but also the system programmer. During implementation they are extremely useful in a kind of bootstrap procedure. Thus the language definition need not have contained any operations at all from the beginning. Only the general concept of procedures is provided in the syntax, as well as the means to generate them. Just for convenience some primitive hardware supported operations like arithmetic operations on integers were, in fact, included. Other operations can be added as desired. In particular all input/output functions as well as most functions to control specific hardware have been realized as external procedures. This technique guarantees a flexible adaption of the system to changing requirements without changes in the dialog language itself.

Default Parameters

In routine applications one mostly uses a set of standard parameters which seldom need to be changed. These parameters could receive pre-defined values. Nevertheless, the option must be preserved to change these parameters in special applications. For that purpose one may assign default values of parameters in the procedure definition, e.g.

```
PROC CONTRAST(I=IMAGE%,LLEV=MIN(I),ULEV=MAX(I):INTEGER)
256X(I-LLEV)/(ULEV-LLEV) → I
RETURN I
```

Variation of the thresholds LLEV, ULEV yield a contrast enhancement of the picture. In a call of this procedure the parameters LLEV, ULEV may be omitted. Then the specified default values will be taken. If explicitly stated, however, they override the respective defaults.

This feature proves to be particularly useful when combined with current variables. In the example the current image is assigned to I as default value. Calls to CONTRAST now can be of the form:

```
    CONTRAST
```

All parameters are used as default from the procedure definition.

```
    CONTRAST(IM,LLEV=10)
```

Only ULEV is used as default.

Dialog Specific Features

The dialog specific facilities are characterized by:

- The concept of *direct* and *indirect* execution as first realized in the language JOSS[11] and adapted by many other dialog systems. Procedures (also called *PARTS*) are created by the user in indirect mode, using the edit facilities of the language, while in direct mode each statement is immediately translated and executed. This concept unifies the usually separate steps of program writing, editing and execution via a job-control language.

- *Interruption facilities* - Interruption of execution may be programmed via a a STOP command or forced by a user interrupt. One can inspect and modify values of variables, execute other programs and finally resume execution.

- *Use of undefined variables* - Variables need not be declared in advance. The user must supply the missing information when the system prompts him. This feature is particular useful in program development and testing.

Image Presentation and Menu Technique

We have paid special attention to image presentation. The hardware contains an (X,Y) CRT display (16 grey levels) with a light pen and a TV display (256 levels of brightness, up to 512 shades of colour) for presentation of x-ray images, with light pen and track ball.

The dialog language offers various means of display control. Objects of any type may arbitratily be added to or deleted from a *Display File*, using the commands *DPLAY* and *DCLEAR*, e.g.

```
DPLAY IMAGE%,'KIDNEY ANGIOGRAM'
```

would issue the current image and the text at the present beam position, while the command

```
DCLEAR IMAGE%(FRAME%)
```

would delete an area defined by the current frame from the image being displayed.

Menu technique is provided by the *DMENU* command followed by one or more items describing the menu, e.g.:

```
DMENU 500,20,LTHRLD,'SUBTRACT LOWER THRESHOLD'
```

Command DMENU itself is not primarily included in the language definition. Rather the language has been extended by this external procedure.

The command issues the text string at position (500,20) on the screen. Pointing to the text with the light pen causes execution of the procedure LTHRLD. Menu creation is greatly facilitated by utilization of structured data types, as the following example indicates:

```
STRUCT MENU=X,Y:INTEGER, EXC:PART, IDENT[20]:STRING
DECL LOWTHR=500,20,LTHRLD,'SUBTRACT...':MENU
DECL UPTHR=500,40,UTHRLD,'ADD...      ':MENU
DMENU LOWTHR, UPTHR,...
```

The first statement declares data type MENU. The two following statements generate the objects LOWTHR and UPTHR. The DMENU command displays these items and leaves the system waiting for light-pen input.

5. Present State

The dialog language has been successfully implemented on a PDP11/45 computer, using the low-level language. To gain experience with the application software for the analysis of x-ray angiograms a stand-alone system PROFI11 [12] has been implemented concurrently.

Presently we insert the algorithms of PROFI11 into the dialog-language system. This is easily performed by using the external procedure technique of the dialog language.

6. Conclusion

The dialog language described is an attempt to provide an efficient, simple and flexible tool for image analysis, especially for medical applications. *Efficiency* of execution and *dedicated hardware* support is guaranteed by combining a high-level interpreter/compiler system with a low-level programming language. *Problem oriented* data types and a versatile display control are adapted to image processing. The needs of the casual user and of routine operation are met by supplying a set of standard functions triggered either by *menu selection* via light pen or by pressing *push buttons*. The dialog technique facilitates production and maintenance of programs by the experienced user.

Acknowledgement

The authors wish to thank Profs. H. Schopper and G. Weber (DESY) for their continuous support of the project and to all members of the DESY-UKE collaboration for many useful hints and discussions.

They are particularly grateful to Profs. W. Brauer and I. Kupka for valuable comments and stimulating discussions.

References

1. K.H. Höhne, G. Nicolae, G. Pfeiffer, W.-R. Dix, W. Ebenritter, D. Novak, M. Böhm, B. Sonne, E. Bücheler; An Interactive System for Clinical Application of Angiodensitometry; Digital Image Processing, Informatik Fachberichte, Vol. 8, Springer, Berlin-Heidelberg-New York, 1977, 232 - 243.

2. G.C. Nicolae, K.H. Höhne; Digital Video System for Real-Time Processing of Image Series; DESY report DV 78/2, Hamburg, 1978, submitted to IEEE Transactions on Computers.

3. K.H. Höhne, M. Böhm, W. Erbe, G.C. Nicolae, G. Pfeiffer, B. Sonne; Functional Imaging - A New Tool for X-Ray Functional Diagnostics; DESY report DV 78/1, Hamburg, 1978, submitted to Radiology.

4. K.H. Höhne, K. Dahlmann, W.-R. Dix, W. Ebenritter, G. Pfeiffer, K. Harm, R. Month; A decentralized Computer System for Processing of Information from Heterogenous Medical Applications; Proc. of the 1st World Conf. on Medical Informatics (MEDINFO 74), Stockholm, 1974, 95 - 100.

5. K.H. Höhne, G. Pfeiffer; The Role of the Physician-Computer Interaction in the Acquisition and Interpretation of Scintigraphic Data; Meth. Inform. Med., Vol. 13, 1974, 65 - 70.

6. I. Kupka; Conversational Languages and Structural Interactive Programming. Formal Languages and Programming (Aguilar, R., Ad.), North-Holland, Amsterdam, 1976, 43 - 64.

7. G. Pfeiffer, K.H. Höhne; A Dialog Language for Interactive Processing of Scintigraphic Data; Proc. of the 4th International Conference on Information Processing in Scintigraphy, Orsay, 1975, 221 - 232.

8. M. Klerer, J. Reinfels (Ds.); Interactive Systems for Experimental Applied Mathematics; Academic Press, New York and London, 1968.

9. G. Pfeiffer; SIMPL11, eine einfache Implementierungssprache für PDP11-Rechner; DESY report DV 76/2, Hamburg, 1976.

10. N. Wirth; PL360, A Programming Language for the 360; Computers Journal of ACM, Vol. 15, 1968, 37 - 74.

11. J.W. Smith; JOSSII: Design Philosophy; Ann. Rev. Aut. Progr., Vol. 6, 1970, 183 - 256.

12. M. Böhm; PROFI11 - A System for Processing and Retrieval of Functional Images; to be published.