stichting
mathematisch
centrum

MC

AFDELING INFORMATICA IW 36/75 NOVEMBER

P.J.W. TEN HAGEN, P. KLINT, H. NOOT & T. HAGEN

DESIGN OF AN INTERACTIVE GRAPHICS SYSTEM

2nd Printing

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.

ACM -Computing Reviews- Category: 8.2, 4.35.

1st printing March 1975

Design of an interactive graphics system

by

P.J.W. ten Hagen, P. Klint, H. Noot & T. Hagen

ABSTRACT

This report contains a set of specifications for a general purpose interactive graphics satellite system. A plan is outlined for the implementation of the system that covers the following subjects:

- The research that has to be done in order to solve the open problems.
- The functioning of the hardware in the system.
- The choice of programming tools and the definition of the modules of the system.
- The organisation of the work to be done, especially with respect to the order in which the various modules will be implemented.

The practical use of the system and research that can be undertaken with the system are outlined.

KEY WORDS & PHRASES: *Interactive graphics satellite system, graphics languages, drawing machines, picture files.*

CONTENTS

CONTENTS

CONTENTS

CONTENTS

0. INTRODUCTION

This report contains a proposal for an interactive graphics system together with a plan to realize it.

The main difference between our proposed system and existing systems is that our system is to be a truly general purpose graphics system. In the first place, this means that it must be possible to attach a wide variety of drawing-machines to the system. To this end, the system will be completely drawing-machine independent except for a basic layer. The basic machine dependent layer should provide a picture file system that allows all system functions to deal with one idealized hypothetical drawing-machine.

In the second place, a wide variety of applications must be possible. The system should support a general purpose programming language for applications and a graphics language of the same (high) level. The graphics language must be capable of controlling the physical machines via the hypothetical machine, provide a data structure for drawings, and allow picture input as simple as its picture output. Furthermore, the graphics language program must share data with the application language program. The solution chosen is to extend a high-level language with data structures and operations for manipulation of pictures.

The first combination of a high-level language with a graphics language will be ALGOL 68 extended in a natural way with graphics facilities. This language called ALGOL 68G will serve as a prototype for similar combinations with other languages.

We have concluded from literature (R.A.GUEDJ [17], F.NAKE and A.ROSENFELD [20]) that similar attempts in the past were not very successful. There are several reasons, however, why we think it is appropriate to start this particular project on computer graphics now.

First of all, we have at our disposal much better programming tools than were available in the past: new programming languages like ALGOL 68 provide the control structure, data structure, and welldefined extendability needed to realize a high-level graphics language. High-level system programming languages delegate irrelevant details to the compiler instead of to the programmer, as was the case with assembly language programming. Moreover, the state of the art in system design is such that considerable insight has been gained in topics like inter-process communication and parallel execution. Present day minicomputers and data communication equipment make it possible to develop satellite systems for special purposes, such as graphics input and output. These subsystems, although comparable in complexity to operating systems for medium size computers, have a relatively simple interface with the operating system as well as with the user (the programmer or the person at the terminal). Moreover, operating systems for these minicomputers exist that are especially well equipped for system software development using high-level languages. Last but not least, new drawing-machines exist that closely resemble an ideal general purpose drawing-machine. These machines follow the general trend in computer hardware of becoming more and more cost effective.

The second reason is that the need for a modest general purpose graphics system (which, according to the plan, will be realized in one of the earlier stages) is growing quickly in the field of programming itself. This is caused by the fact that recently developed programming languages (general or special purpose) allow manipulation of complex data structures. These data should be drawn rather than printed in order to visualize them for the user. This is true for input as well as output. The classical I/O devices require linearization of essentially two-dimensional data. This situation constitutes a serious drawback in all systems that support the languages mentioned above.

The existence of better tools does not mean that all problems are now solved. In particular, the interactive use of a graphics system is still an open problem (see P.BOULLIER [21] for an advanced general purpose interactive system which allows local interaction only). Chapter 1 of this report describes a number of research subjects we will have to deal with, more or less extensively, in the course of our project.

Chapter 2 describes the hardware configuration and the so-called "hard" software on which the system must be implemented.

Chapter 3 contains a functional description of the graphics system. This inventorization, together with some design concepts, leads to a division of the system into modules that can be implemented independently.

Chapter 4 is a concrete implementation proposal for the system of chapter 3 on the hardware of chapter 2. The choice of the programming tools and the way they are applied receives special attention.

Chapter 5, finally, contains a plan for the organisation of the work to be done. The order in which the various modules are to be implemented is defined. Several milestones are placed along the road that leads to the final system. Each milestone corresponds to a new facility that will have been added to the existing part. Together with the description of the work required for each module, time estimates are given based on the assumption that the work will be carried out by a team consisting of the four authors of this report. It is concluded that the complete system will require a team effort of at least four years.

1. INTERACTIVE GRAPHICS SYSTEMS AND RESEARCH

1.1 Interactive graphics systems

An interactive graphics system is defined as an operating system that allows input and output in the form of a picture at run time.

Graphics systems can be divided into general-purpose and special-purpose systems. In the second type of system, programs can only use graphics I/O in a limited field of application (aerodynamic calculations, electrical networks, etc.). One of the problems in general purpose systems is that input to programs must be possible in a uniform way for a wide variety of applications and all kinds of different drawing-machines.

A second problem is the definition of an interface between a graphics language (even if elementary) and a high-level programming language. One way of overcoming this difficulty is to represent the graphics elements as procedures in the high-level language. This leads to the display procedure approach, as used in various FORTRAN-based graphics systems (see D.GROOT [15], A.D.RULLY [26]) or ALGOL-like systems (see W.M. NEWMAN [22]). The nesting structure of the procedure calls and declarations used can be a reflection of the structural properties of a picture. This situation imposes two serious limitations on interaction, however:

- The structure of nested procedure calls cannot be changed interactively.
- The display files which are the output of the graphics programs are unstructured sequential files.

We intend to avoid these limitations by defining our graphics language as an extension of ALGOL 68. In this way the interface between graphics and nongraphics language elements is completely defined within the ALGOL 68 framework, without the need of compiler modifications. The data structuring possibilities of ALGOL 68 will provide the display file structuring necessary for interaction. The ease of defining and implementing graphics languages in ALGOL 68 makes this method attractive even when the ultimate goal would become a graphics language implemented on a completely different host language.

1.2 Research aspects

The research aspects of the graphics project can be divided into:

- Research during the design of the system.
- Research carried out with the help of the system.

Of course, the usefulness of the system can be judged from the success of research projects of the second type. Such an evaluation may even lead to changes in the initial system. The same remarks can be made for research during the design phase, when already implemented parts of the system are applied. Hence the two research categories mentioned, are not really disjunct. For convenience however, we will adhere to this distinction.

1.2.1 Research in the design phase

1.2.1.1 Programming tools

Programming tools will be understood, in this context, to be languages and language elements used for the implementation of the graphics system. In this section we will discuss the programming tools whose design or selection requires some further research.

The major system programming task consists of the implementation of an interactive graphics terminal on a satellite computer. To this end we need an operating system that supports a system programming language. The operating system chosen at the start will be UNIX (see D.M.RITCHIE and K.THOMPSON [24]), which supports the high-level language C (see D.M.RITCHIE [25]). This language does not contain primitives for system programming however, like semaphores, critical regions etc. An alternative might be the use of ALEPH (see D. GRUNE et al. [16]), extended with system programming primitives. ALEPH has a drawback: no implementation on the PDP11/45 is available.

Some associated projects are selection of programming primitives and extension of either ALEPH or C. Primitive actions will be designed with the help of simulation in the candidate languages. In the experimental stage they can be changed without the necessity for regular compiler modifications. Noninterpretive implementation can follow afterwards.

The host language used for the first implementation of the high-level graphics language will be ALGOL 68. Therefore the constructions in this language are potential programming tools in the sense used here. The choice of language elements to be used for the implementation of the graphics extension constitutes another design problem. For example, unlike the proposal of DEHNERT ao.[7], the mode "flexible row of ..." will not be used to implement the union operator, to avoid unnecessary copying operations.

1.2.1.2 Intermediate language

The IL (Intermediate Language) will be discussed in detail in section 3.3. For the moment, it suffices to know that the IL is an intermediate level language between high-level graphics languages and the machine languages of graphics I/O devices. It is a language for the definition of pictures. In our system, a high-level graphics language that generates a picture will do this in the form of an IL program. This program can be stored in a picture library or translated to the machine code of an actual graphics device. When pictures are read in from a graphics device, the reverse steps are taken. This results in an IL picture description that can be mapped into a data structure available to the high-level graphics program. In our view, the design and implementation of the Intermediate Language presents the following research problems:

- To design IL primitives for storage and retrieval operations in picture libraries. These primitives must enable the description of pictures containing references to library subpictures.
- To find a level of complexity between the high-level graphics languages and the drawing-machine languages that allows efficient conversion in both directions.
- Efficient conversions to and from the Intermediate Language are of particular importance: on the one hand, they influence the execution speed of complex library manipulations and on the other hand, they influence the execution speed of all transports between program and device.
- The construction of an abstract machine to be used for the definition of the semantics of the IL.

1.2.1.3 Primitive graphics elements and data types

An important subject for research is the design of a graphics language. First of all a set of graphics primitives has to be defined, i.e. primitive data types and primitive operations. Next, constructions for composition of complex data types and operations on complex data have to be built. These language elements can then be used to give structured descriptions of pictures together with their associated nongraphic information. An example of such a language is the design of the ALGOL 68 extension discussed in section 3.3 and TEN HAGEN[29].

The importance of this aspect of the design of the interactive graphics system is self-evident. The appearance of the system as seen by the user is strongly determined by the facilities offered in the high-level language. All facilities provided should be available through the high-level language.

1.2.1.4 Interaction

The study of interaction mechanisms is another important research project during the design of our graphics system. It is known in principle how to construct or modify complicated drawings on a display with the aid of light pens etc. The problem of supplying those drawings as input to a program has not been solved satisfactorily. In fact, this would require a form of pattern recognition which enables the generation of a data structure of a type known to the program, on which the drawing can be mapped.

Interaction with programs in present day general purpose graphics systems proceeds, as far as we know, in the following two steps:

- Parts of the drawing that must be detectable by a light pen, have to be identified as such statically.

- After a light pen hit on such a part, a simple action like deletion, replacement or transformation, is undertaken. This action is either predetermined or selected from a limited set with the help of, for example function buttons.

In this type of interaction, the data structure representing the picture can be changed only by inserting or replacing subpictures known in advance. The input of arbitrary pictures, constructed at a display, is very difficult. The situation in three dimensional computer graphics seems at first sight somewhat different due to the existence of transformations such as hidden line algorithms, which drastically change the picture displayed. There are two data structures involved here: one representing the three-dimensional picture, the other representing a two-dimensional projection of the same picture. The former is not changed by hidden line algorithms and the like. In general it cannot be changed interactively from a display because of the irreversibility of projection transformations. Furthermore, there are to our knowledge no general purpose graphics systems in which the two-dimensional projections can be used as program input. This problem is of the same complexity as the input of arbitrary pictures.

We will start with the investigation of an interaction mechanism that is characterized by the following properties: picture elements are light-pen detectable if and only if they are declared to be so called "basic"- or "structured" symbols. In general, any (sub)picture can be declared to be a basic symbol or a structured symbol. Basic symbols are, apart from graphics primitives like lines, points, etc., picture elements whose internal structure is reduced to the simplest form possible, namely a list of primitives. The data structure of a structured symbol remains identical to that of the picture. Symbols can be defined both statically and dynamically. Because of the fact that symbols are the only picture elements that can be referred to during interaction, those defined dynamically must be composed from symbols already existent. In this way, the data structure representing a drawing

can be altered drastically, for instance, by replacing an old symbol with a newly defined one. Conflicts between interpretations of light pen hits will be resolved by supplying the type of the structured symbol pointed at. An important aspect of symbols will be the possibility of defining them as pictures with associated nongraphic data. This will increase the usefulness of picture input to programs. How this association will be accomplished still has to be investigated.

A further aspect of our interaction mechanism will be the implementation of a specially designed flow of control language that is part of the conversation between the user and a running program.

A few words about a possible realization may be appropriate here: the places in a program where interaction has to take place could be indicated by calling a procedure "interaction". This procedure could be the standard one or may be user-defined with the help of primitives provided by the system. It can process the input generated at a display and produce the necessary changes in the data structure of the picture. All information passed between the high-level language program and a display is converted to an IL program (see 1.2.2.2). This introduces the problem how a symbol identified at a display can be found in the IL program and in the high-level data structure. These problems strongly resemble each other. We suggest two solutions. We will use the abbreviations DM and HL for drawing-machine language and high-level language in this discussion.

- There exists a table of pointers to all symbols in the picture data structure. This table is produced during the conversion between either IL and DM or between HL and IL, and is added to the target language file. The procedure "interaction" and its analog from DM to IL use these pointers to locate a symbol in the data structure.

- Some information on the position of the symbol in the picture is transmitted to the interaction mechanism which searches the symbol in the HL or IL data structure by scanning it, directed by this information.

The mechanisms just described are mentioned only to show, that our interaction proposals can be implemented. The actual mechanism we will use remains to be investigated in detail.

1.2.2 Research with the graphics system

1.2.2.1 Display of complex information structures

Complicated structured information can best be visualized in the form of pictures, when it has to be used by human beings. This fact suggests using a graphics system as a programming aid by having it draw data structures and flow charts. In particular, we might think of the following examples, each of which could be a more or less extensive research project:

- The data structures that exist during the execution of an ALGOL 68 program can be represented in the form of a directed graph. Such graphs could constitute important diagnostic information.
- Monitoring and displaying the flow of control in programs at run time. This could be used both as a debugging aid and as a design aid, in particular in the study of complicated programs like operating systems.
- The visualization of statistical information about the runtime behaviour of programs, like calling-frequencies of subprograms, resource usage, et cetera.

- Control of input to programs at a visual display, in those cases where the correctness of the input can best be verified by means of a picture.

1.2.2.2 Typographical aids

At the Mathematical Centre, a lot of work has already been done on the development of various programs that perform typographical functions. For further information see the list of references in TEN HAGEN [30]. With an interactive graphics system, research in computer typesetting can be continued. We merely list here some of the possibilities:

- Interactive programs for the determination of the layout of complicated mixtures of text and illustrations.
- Interactive text editors that can cooperate with the layout programs.
- Interactive programs for generating special characters.
- Programs for generating tables.
- Specialized programs like music editors.

1.2.2.3 Classification and recognition of pictures

An important classifier in information storage and retrieval systems for two-dimensional information is a linear key appended to each picture. This key is a linear character string, which can be used as an index. The program can interpret the key as a schema for the picture. This simplified picture can be presented to the user as a notation for the key. An example of such keys is the Wiswesser line formula chemical notation (see E.G.SMITH [27]). This notation is strictly based upon the topological properties of chemical structure formulas. This kind of classification can be used in question-answering systems for pictures that deal with both their topological properties as well as properties

specified by associated data.

The first step in picture recognition is to define the notion of recognition. Such a notion can be obtained by specifying all (possibly not disjoint) sets that a picture might belong to. When classifications with the help of linear keys (see above) are available, research can be done in the development of algorithms that produce these keys from input drawings.

Another picture classification that seems worthwile to investigate is based on the data structure that represents the picture. In this way, pictures processed or produced by a program can be classified at the same time.

2. CONFIGURATION

The Interactive Graphics System (further referred to as IGS) will be implemented on a network of three computer configurations, namely: a CYBER73-28 (CDC), a PDP11/45 (DEC) and a PDP8/I (DEC). The PDP11/45 will function as a graphics satellite of the CYBER73-28. The CYBER will execute the graphics application programs. All interactive graphics devices are connected to the PDP11/45. The PDP8/I will function as a data concentrator for the PDP11/45. Thus, logically, all peripherals can be considered as connected to the PDP11/45.

For users of the CYBER73-28 it will be possible to produce files containing graphics information. These files can be put in the output queue for the hardcopy graphics devices of the satellite. In a later stage this facility will be extended to enable the (interactive) production of picture files at the graphics terminal for input to application programs running on the CYBER73-28.

Our main programming task consists of the design and implementation of the complete software package for the PDP11/45 graphics satellite.

The description of the hardware configuration in this chapter serves two purposes:

- A hardware overview in order to illuminate its usage in graphics applications.
- Specification of the programming of this hardware on the lowest level.

The discussion of the CYBER73-28 is restricted to the interface with the satellite. A survey of the PDP11/45 - PDP8/I configuration is given in fig 2.1.

(Just prior to the publication of this report we decided to use a 16K PDP8/E instead of an 8K extension of the PDP8/I. The RK08 disk unit is connected to the PDP8/E. The PDP8/E is connected to the PDP11/45 in an identical way as the PDP/I. The PDP8/I and the PDP8/E are also connected by means of an interface designed by P. BEERTEMA. This change in configuration does not affect any conclusion in the report.)

2.1 CYBER73-28 configuration

The CYBER73-28 computer (2 processors) at SARA provides on-line terminals and batch terminals. In the network the CYBER73-28 will treat the PDP11/45 as a remote batch terminal. The operating system SCOPE [4] provides features which are necessary in our IGS, such as high-level languages and file support. Furthermore the users of the CYBER73-28 will have the opportunity to display picture files on the graphics devices of the PDP11/45.

Initially the communication will conform to the SARA-implementation of the Mode II protocol [3], which supports streams only for card readers, card punches, printer and an operator console. Conversion for graphics devices will have to be done at the satellite. Installation of an interactive stream will require an extension of SCOPE in order to support interactive streams in the Mode II protocol.

2.2 PDP8/I configuration

2.2.1 Hardware

The PDP8/I computer is equipped with 24K words of core and a memory protection unit. It has to be seen as a data concentrator for the PDP11/45. The I/O devices of the PDP8/I are:

- Background memory: a dual dectape unit, a disk unit, and a disk cartridge unit.
- Graphics devices: a storage display unit with a joystick and a small Calcomp plotter.
- Terminal devices: a console teletype and two Olivetti terminals multiplexed to the PDP8/I.
- Converters: three D/A converters.
- Interface device: an interface for the connection with the PDP11/45, designed at the Mathematical Centre by P. BEERTEMA.
- Miscellaneous devices: two paper tape readers, two paper tape punches, a line printer, and a braille printer.

For specific details about the devices see table 2.1.

2.2.2 Function in the IGS

The purpose of the PDP8/I configuration with respect to the IGS is as follows:

- Support of programming tasks on the PDP11/45: the background memory for file storage (permanent storage on DECtape and paper tape, and temporary storage on disks), and the line printer for assembly listings and editing facilities.
- Servicing the I/O devices under the Mode II protocol.
- Integration of the KV08 display and the Calcomp plotter in the IGS.

The existing programs on the PDP8/I are already capable of performing most of these tasks under the operating systems TRACK, OS8 and UTOR. The remaining tasks will be carried out as part of a different programming project. For completeness we list here the programming tasks in this project which are required for the IGS:

- Implementation on the PDP8/I of the communication protocol (see HAGEN [18]) between the PDP11/45 and the PDP8/I.
- Extension of the single user editor to a multi user editor.
- Some conversion routines to and from Mode II format.
- Integration within the IGS of the plotter and the KV08 display (it has not yet been decided whether this will be carried out by us or others).

2.3 PDP11/45 configuration

The PDP11/45 computer will initially be provided with 48K words of core, a memory management unit and a floating point processor. See the DEC handbooks [10] and [11].

2.3.1 General purpose devices

2.3.1.1 Background memory

There are two disk cartridge units, one for system working space, and the other for graphics data files.

2.3.1.2 Console

The DECwriter will be the standard operator console. The GT40 can also be used for this purpose if a GT40 console driver is written.

2.3.1.3 Data communication equipment

- CYBER73-28 connection. The PDP11/45 is initially connected to the CYBER with a 9600 BD synchronous telephone line, later to be replaced by a 50 KBD videoline. The PDP11/45 equipment consists of a DQ 11-AE [8] with hardware cyclic redundancy check, programmable character recognition and direct memory access. Communication has to be maintained according to the so called Mode II protocol [3]. This fixes the way the DQ 11-AE has to be programmed. The program for the protocol on the level above the DQ 11 driver, will be taken from the DEC COMTEX Mode II ISR/TAB [12]. The exchange of information with the protocol will be on a file by file basis. This will imply file conversion in most cases.
- PDP8/I connection. A DR 11-C unit provides a full duplex 16-bit-parallel data transport. The DR 11 handler takes care of the data transport according to the PP protocol [18]. The communication with the PDP8/I is on a file by file base. There is no master-slave relation, both participants in the communication behave symmetrically.
- GT40 connection. DL 11-E modules (see [11]) take care of a full duplex 9600 BD serial data transport between the PDP11/45 and the PDP11/05, which is part of the GT40. The GT40 has no storage capacity for complete files. Information must be exchanged below the file level. The communication protocol still has to be designed.

2.3.2 Graphics devices

2.3.2.1 HRD-1 Laser Display/Plotter

The HRD-1 of Laser Scan Ltd. (see [19]) is a high resolution film recording and image display system. The HRD-1 can be used for drawing in three ways:

1. High quality hardcopy output on diazofilm.
2. Output to a large screen in storage mode. The storage mode is obtained by recording the picture on semi permament photochromic film, which is projected on the screen.
3. Output in refresh-mode directly on the screen.

The refresh-mode picture is superimposed on the storage-mode projection. Storage of the refresh mode picture is avoided by selecting a different address area. This constitutes the only difference between storage mode and refresh mode. The writing beam (from an argon ion laser) is successively controlled by the following four components: a modulator, a secondary deflection system, a dynamic focussing lens and the main deflection system. The accuracy and high resolution are obtained by an autonomous interferometric control in the main deflection system. An auxiliary laser supplies light for the interferometer. Program control of the writing beam is obtained by feeding the four components through the HRD-1 computer interface. Sixteen function buttons, a tracker ball and a keyboard provide the HRD-1 with interaction facilities.

Computer control of the HRD-1 is provided through an interface. Sixteen PDP11/45 memory registers are reserved for the control functions and the status and data information. The control functions are:

- Global and local moves.
- Mode-setting and focussing of the beam.
- Film control.

Status and data information specify:

- Machine status, interrupt flags and error conditions.
- Position of the beam.
- Function buttons.
- Keyboard codes.
- Position of the tracker ball.

The HRD-1 will be used for hard copy output on film of batched picture and text files, with the emphasis on pictures, and for interactive graphics terminal sessions.

2.3.2.2 GT40 display

The GT40 interactive graphics display consists of a PDP11/05 processor with 8K memory and a display unit. For interaction the GT40 has a keyboard and a light pen. The screen, a cathode ray tube, is only 17.1 by 22.8 cm. It will be used for the following purposes:

- Inspection of picture files.
- Simulation of other graphics devices.
- Interactive experiments. The GT40 is relatively easy to program, and therefore well suited for graphics experiments.
- The development of special character sets for specific applications and special hard-copy output devices, like the Digiset and the HRD-1.
- To display information generated by tracers that monitor (system) programs (see 1.2.2.1).

- To use the GT40 as a character display.

All the software for the GT40 has to be developed. At the basic level it performs the following tasks:

- The interface with the PDP11/45: design and implementation of a communications protocol.
- The extension of the display-processor language with virtual instructions in order to obtain a more manageable display.

2.3.2.3 The Digiset 40T2

Although our configuration does not contain a Digiset, the possibility exists to use one. Connection with this Digiset can be established through a telephone line or via a magnetic tape unit. The Digiset 40T2 [14] is a photo-typesetter that can be computer-controlled. The photo-typesetter produces hard-copy output on film or paper. Input consists of 8-bit transparant code supplied through a paper tape reader, a magnetic tape unit or an on-line computer connection.

The Digiset can operate in two modes, namely, the character definition mode and the typesetting mode. Switching between the two modes is under input-code control and can be done at any moment during operation. Special properties of the Digiset with respect to character definition and positioning allow application as a drawing-machine of high precision (0.01 mm) and acceptable speed (100 times faster than a plotter).

The best results of the drawing capacity can be expected for pictures that combine drawings and texts, with emphasis on texts. The text-oriented design results in a considerable better performance when the overall movement of the beam is left to right.

The application as a line-drawing-machine requires considerable programming effort in addition to the substantial effort to use it as a text-composer (see H.NOOT and P.J.W.TEN HAGEN [23], P.J.W.TEN HAGEN [30]).

The Digiset can use only program-defined characters of arbitrary shape. These character sets are organised in type founts and stored fountwise on disk. By (re)defining characters dynamically as needed, an in principle unlimited character set for texts, line drawings and grey-scale pictures is available.

PDP8/I configuration tabel 2.1

device description	number	name or ser.nr.	manufacturer	remarks
dual dectape unit	1	TU 55	DEC	
disk unit	1	RS08	DEC	256 K words
disk cartridge unit	1	RK08	DEC	1.2 M words
storage display	1	KV08	DEC	
Calcomp plotter	1	565	Calcomp	300 st/s,30 cm
console teletype	1		Teletype Corp.	10ch/s
Olivetti terminal	2	TE 318	Olivetti	110bd 80c/line
D/A converter	3	AA-01	DEC	
PDP11/45 interface	1	D1 811/I	MC	
paper tape rd/punch	1	PC 04	DEC	300 c/s,30 c/s
paper tape reader	1	EL 1000	Electrologica	1000 c/s
paper tape punch	1	Facit 1	Facit	150 c/s
line printer	1	Anelex	Anelex Corp.	20 lines/s
braille outputdevice	1	DNL		7c/s

PDP11/45 configuration tabel 2.2

device description	number	name	manu-fact.	remarks	intern ref.	extern ref.
PDP11/45 processor	1	KB11-A	DEC		2.3	proc.handb.
floatingpoint processor	1	FB11-B	DEC		2.3	proc.handb. chapter 7
clock	1	KW11-L	DEC	timeslices of 20 msec	2.3	per.handb. page 4-197
16K core	2	MF11-UP	DEC	with a controller	2.3	per.handb. page 4-245
16K core	1	MM11-UP	DEC		2.3	per.handb. page 4-245
memory manager	1	KT11-B	DEC		2.3	proc.handb. chapter 6
bootstrap loader	1	MR11-BB	DEC		2.3	
DEC writer	1	LA30-CD	DEC	console writer	2.3.1.2	per.handb. page 4-201
interface to DEC writer	1	DL11-A	DEC	300 BD	2.3.1.2	per.handb. page 4-124
interface to CYBER	1	DQ11-EA	DEC	50 K BD, cyclic redundancy check char. recognition	2.3.1.3	DQ11 NPR synchronous line interf. (preliminary) 1974
interface to PDP8/I	1	DR11-C	DEC	16 bit parallel, 250KBD	2.3.1.3	per.handb. page 4-165

interface to GT40	1	DL11-E	DEC	9600BD ser. asynch.	2.3.1.3	per.handb. page 4-124
null modem	1	H312 A	DEC	for interf. to GT40	2.3.1.3	per.handb. page 4-185
disk controller	1	RK11-D	DEC	controller (two disks)	2.3.1.1	per.handb. page 4-282
disk cartridge drive	2	RK05-BB	DEC	1.228 m words	2.3.1.1	per.handb. page 4-282

proc.handb. = processors handbook PDP11/45, 1974-75, DEC
per.handb. = peripherals handbook PDP11 , 1973-74, DEC

GT40 configuration tabel 2.3

device description	number	name	manu-fact.	remarks	intern ref.	extern ref.
PDP11/05 processor	1	KD11-B	DEC		2.3.2.2	proc.handb. chapter 4
keyboard	1	LK 40	DEC		2.3.2.2	GT40 users guide page 6
interface to 11/45	1	DL11-E	DEC	9600 BD serial	2.3.2.2	per.handb. page 4-124
8K core	1	MM11-L	DEC		2.3.3.3	per.handb. page 4-245
GT40 display and processor	1		DEC		2.3.2.2	GT40 users guide

proc.handb. = processors handbook PDP11/05, 1973, DEC

per. handb. = peripherals handbook PDP11, 1973-74, DEC

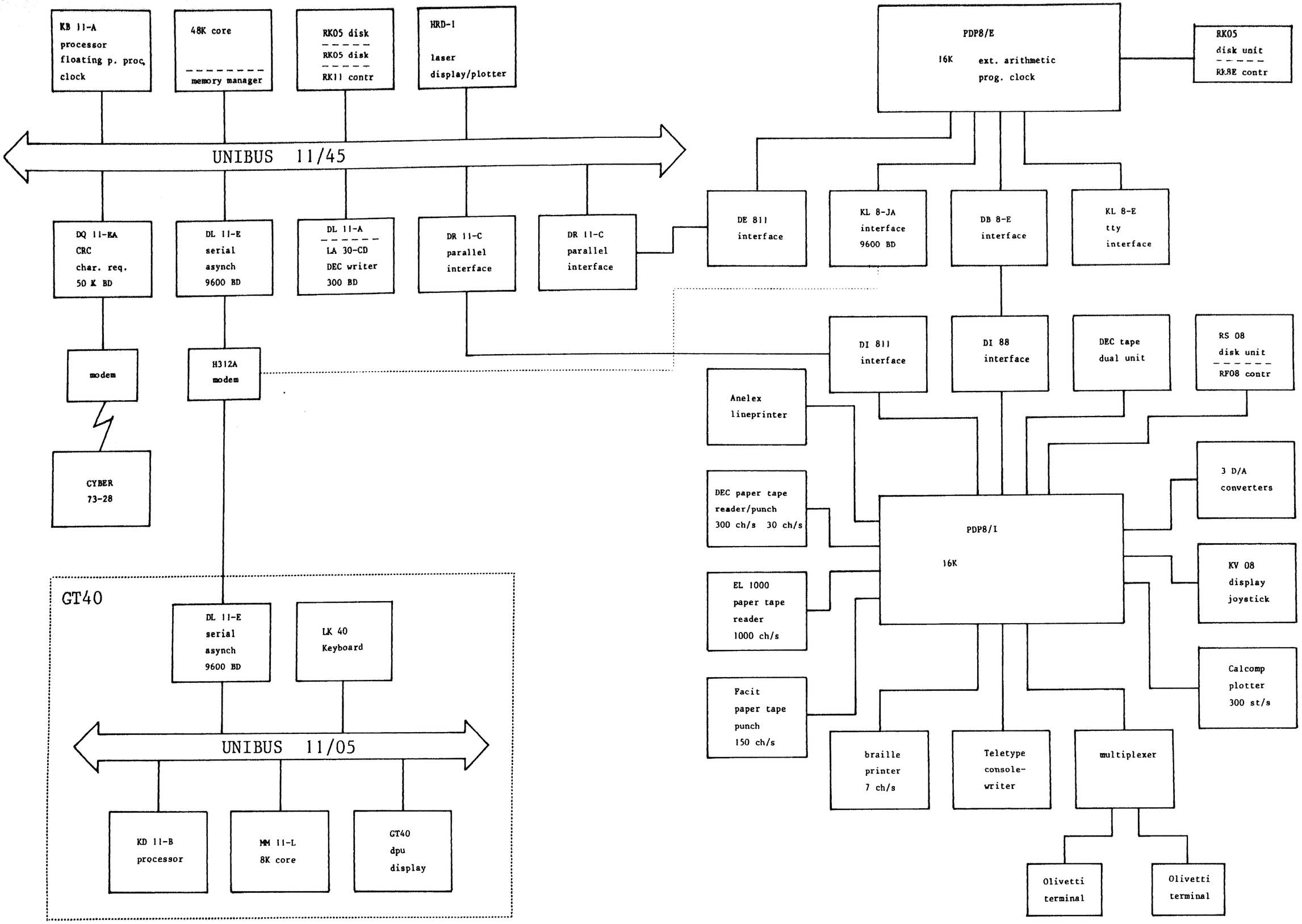


fig. 2.1. A survey of the PDP11/45, PDP8/I and PDP8/E configuration

3. PROPOSAL FOR AN INTERACTIVE GRAPHICS SYSTEM

3.1 Overview

In this overview we will split up the overall function of the IGS in three basic functions. Each graphics program will require execution of all basic functions.

The first basic function is to execute graphics application programs. To this end the IGS supports high-level graphics languages. The effect of a graphics application program is to read and to produce zero or more picture files.

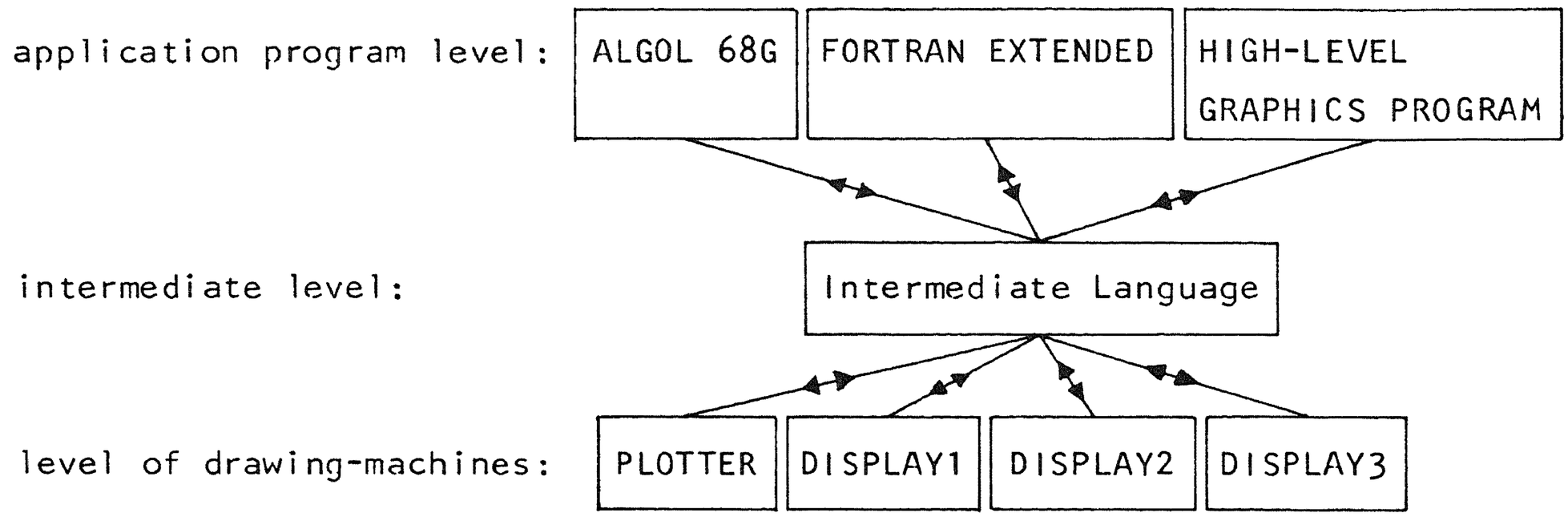
The second basic function is to drive all physical drawing machines. The drawing-machines either produce pictures in hardcopy (paper, film) or on a screen, or they produce picture input in the form of a picture file.

The third basic function is to provide a link between the two previous tasks. This linking function is realized in the form of a picture file system. All picture files contain pictures in a standard internal representation, the so-called Intermediate Language (IL).

A typical graphics session encompasses the following steps:

- A graphics language program is compiled and brought into execution (function 1).
- The program opens a number of picture files. Open files can also be linked to a drawing-machine, for input as well as output (function 3).
- The user starts the drawing-machines. All input is put into files that will be consumed by the application program. All output files for a drawing-machine are converted and consumed by the machine-driving program (function 2).

level overview fig. 3.1



The proposal in this chapter will be formulated by describing the three basic functions in more detail. The main strategy is to split up the functions into subfunctions that can be realized (implemented) as more or less autonomous subsystems (modules). Each module is characterized by its functions, its relations with other modules and its internal structure. Before describing the modules a few design decisions that apply to all functions are outlined.

The most important decision is to formulate the various functions with the help of special languages. These languages will either be implemented directly (e.g. the high-level programming languages) or will be used only to guide the implementation (e.g. the IL).

A second decision deals with the information flow between the various functions. The main goal here is to achieve efficiency by finding a uniform way of exchanging information.

The third decision concerns the definition and realization of interaction. It will turn out that a two-level interaction mechanism is required.

3.2 Language Aspects

In each of the three basic function modules one or more programming languages can be distinguished.

In the application module there exist several high-level graphics languages. Each general purpose programming language (e.g. ALGOL, PASCAL, FORTRAN) can be extended to a graphics programming language, in a more or a less convenient way.

In the drawing-machine module there exist various drawing-machine languages (machine instruction repertoires).

The linking function contains only one language, the so-called Intermediate Language (IL). This minimizes the number of conversion routines needed for translation between high-level graphics languages and physical machines: one for each language and one for each machine (see fig 3.1). The IL can also be used for conversion between pictures from one drawing-machine to another (e.g. screen to plotter).

The IL is a language designed only for internal representation of pictures. Its semantics can be described with the help of a abstract drawing-machine. The graphics I/O of the high-level graphics program consists of IL programs. The high-level program must be able to generate and read IL-programs. The graphics I/O-instructions consist of conversions between IL-programs and their representation in the high-level graphics program. This conversion can be made efficient by embedding the IL in the high-level language. This also provides the programmer with one conceptually simple drawing-machine (that is if we can invent a simple IL). In fact we intend to realize all high-level graphics languages by embedding the IL in an existing general purpose programming language.

For both input and output we want to use the same IL. This guarantees complete symmetry between input and output at the application level, which again will promote simplicity.

In order to obtain a flexible transfer system between IL files and the various drawing-machines, it may be appropriate to proceed in steps. This means that a drawing-machine is transformed into an extended machine by adding instructions. The extended machine should resemble the IL machine more closely in order to simplify conversion. The process can be repeated as often as needed.

This strategy is indispensable for the realization of drawing-machines for input. The distance between a light pen hit and any IL primitive is much greater than the distance between a vector function on a screen and a line in the IL. It is at this point, rather than at the application level, that the problems involved in obtaining symmetry between input and output must be solved.

3.3 Principles of Interaction

Interaction is a special combination of input and output. Input provided by a user goes directly to a program that is in some form of waiting state. Output goes directly to the user at the terminal. Input and output are exchanged on a question and answer basis.

Graphics interaction means that both input and output are pictures. All input originates from a picture produced at a screen (visual display). Output pictures are drawn on the same screen.

From this description, it follows that the interactive part of a graphics system performs four subfunctions:

- Picture input.
- Picture output.
- A question and answer mechanism.
- An arbitration function for sharing the screen.

In general, after a question by the program, the user needs the complete drawing-machine for composition of the picture that constitutes the answer. He also needs to reinspect the question picture from time to time. Hence the question must be redisplayed and therefore remembered by the system. This can be accomplished in two ways. The user can ask the application program to repeat the question, or the question can be redrawn from local storage. The latter solution requires a local file capacity. We nevertheless prefer this solution because it strongly reduces overhead and local file capacity will be necessary anyway.

For composition of a picture on a screen one can again choose between two alternatives. In the first approach, user actions can be recorded in a sequential file and be sent to the program. The program must inperpret this raw material in order to extract the intended picture. It would be wise to display this picture at the screen and allow some additional composition in order to ensure a correct answer. Most existing graphics systems apply this raw picture input, usually without the correction cycle. We strongly prefer a different approach for the following reasons:

- Raw input violates the concept of symmetry between input and output, since the primitive actions at the screen are more primitive than the primitives in the IL. (for instance, a light pen hit or a function button push is not a point or a line, at best a point or a line can be the result (composed) of a sequence of hits and pushes).

- Verification of the right input should be possible through a local drawing- and editing system. The activation of a big application program for input that will instantly be rejected, causes an unacceptable overhead.

The second approach uses a local picture editing system. A picture editor is a much more complicated program than a text editor. A text is a linear string of characters that is built up on a line by line basis. As a consequence only the current line is subject to changes. Therefore in most text input systems a local mode is used only to prepare a line of text, which will be handled over as one information item. As compared to texts, pictures require at least one more level of interaction. The current picture input devices require a series of input actions in order to specify one input primitive. Such a series can be compared in length and complexity with one line of text. Pictures, however, have a tree-like information structure. This means that such a primitive must be linked to several points in the already existing picture. This linking (composing, associating, constraint building, etc.) will require a new series of input actions that may again be subject to local mode editing. Furthermore, the complete picture must remain available in order to specify the linking points.

The corrections in the picture must be possible in terms of primitives (deletion, replacement) and by relinking these corrections to the original picture. This editing on the level of primitives and links is what we call the local interaction mode. The more primitive interactive mode of building picture primitives and link specifiers is not considered as an extra interactive level (although it is). We expect no more difficulties with it then we would encounter while collecting lines of text.

Complete pictures thus constructed constitute the input to the interactive application program. This high-level interaction or global interaction introduces two new problems in an IGS. Both have been discussed already in chapter 1 as research subjects. We do not want to repeat that discussion here, but instead try to define functions of the system that would be able to solve these problems.

The careful reader will have noticed that we let the program ask the questions, and the user provide the answers. This is because answering requires much more intelligence than asking. When asking a question the program must also specify the form of the answer with respect to:

- The level of detail of the answer. In the exchange of tree-like information it is generally convenient for the user to see as much of the tree as possible, although only part of the tree is subject to discussion. A special selection module should help to extract the subpicture that constitutes a permissible answer out of the picture supplied by the user.
- The topological (grammatical) properties of the tree as a graph. There must be a function module that will reject all picture trees of the wrong type. In this way the program can protect itself against dangerous input (for instance, it can specify the branching depth).

The first function can be characterized as defining dynamically the permissable actions. It is invoked prior to answering. The second function can be characterized as checking the permissable structure of the answer, and is invoked after the user has declared that the answer is ready.

The question and answer mechanism is a monitor function responsible for the conversation dialogue. It decides whose turn it is and at which moment turns change. It should provide both user and application program with simple rules for conversation. All violations of these rules should be caught by this mechanism. As a result, the module manages the drawing-machine as a resource shared between user and program.

3.4 The picture file system

The picture file system performs the linking function between the application level and the device level.

The interfaces with the other two basic modules consist only of conversion routines to and from the Intermediate Language representation of picture files and file operations.

The information exchange in the graphics system basically consists of picture files. To this end all communication links must be able to transport complete files. The basic layer that creates this facility will not be discussed here. We merely assume that a file transporting system exists. For the realization of this layer we refer to the paragraphs on protocols in chapter 2 and 4 of this report. The file operations we have in mind are the well known file handling operations present in most operating systems (e.g. open, close, read, write etc.). It follows that all information which has to be passed between the application level and the drawing-machines must be expressed in the IL. This information deals with the following subjects:

1. The application program must have complete control over all drawing-machines.
2. A tree structure must be present in the IL in order to preserve the relations between subpictures.

3. The IL must be able to specify interaction patterns.
4. A pointer mechanism or another selection mechanism must be provided that identifies a particular piece of a data structure representing a picture in the application program. The user makes his selection based on the picture displayed on the screen.
5. Restrictions on the form of IL programs must be expressible with the help of other IL statements in order to protect an application program from wrongly structured input (e.g., grammatically wrong IL programs or too deeply nested subtrees).
6. Library functions must be provided for selection of pictures and sublibraries, and for building up libraries of characters and pictures.

At this stage, a draft proposal for an IL exists (see P.J.W. TEN HAGEN [29]) that covers the subjects 1 and 2 with respect to line-drawing-machines. In chapter one of this report it is pointed out that mechanisms exist for subjects 4 and 5. The interaction and library functions still have to be designed.

The linking module sees every drawing-machine as an IL machine. Conversion routines exist between each physical device and an IL file. A one-to-one mapping of such files on devices is possible. The conversion routines are considered part of the machine-driving module. The user-program can manipulate devices as if they were files. The same reasoning holds for the sharing of a device between user and program during interaction. This monitoring function will be carried out by the so-called Interaction Monitor.

A second important function of the linking module is the local editor (IL file editor) for picture preparation and correction. This editor has a number of internal links to the Interaction Monitor:

- It depends on the Interaction Monitor for the switching from active to passive state during interactive sessions.

- All access to symbol tables and picture libraries goes via the Interaction Monitor.
- The Interaction Monitor can initiate a new edit cycle when the form of the input violates the restricions defined in the previous questions.
- All file management is passed to the file manager through the Interaction Monitor.

3.5 The application level

In the application level we find all facilities that are common to general purpose operating systems. We will consider only those aspects that are typical for a graphics system.

3.5.1 High level graphics languages

As can be concluded from the description of the linking function, especially from the role of the Intermediate Language, all facilities of the graphics system can be controlled via the IL. The most important design decision we have made is to provide these facilities to the user by embedding the IL in a high-level programming language.

This approach has in our opinion a number of advantages, which are stated elsewhere in the report in different contexts. We repeat them here for completeness:

- It minimizes the number of primitive concepts, since the application level deals with only one drawing-machine (the IL machine).
- Existing high-level languages provide lots of features that are useful for a graphics language.

- The application program as a whole and the graphics part of it share the same data structure (of the high-level graphics language).
- Recently developed general purpose programming languages claim to allow this kind of embedding (for instance ALGOL 68). We intend to take this gauntlet thrown down by ALGOL 68.
- A well defined high-level language is a prerequisite and a starting point for a well defined graphics language.

In TEN HAGEN [29] a proposal is given how to embed the IL in ALGOL 68. The extendability of ALGOL 68 allows this type of embedding without any modifications of the ALGOL 68 compiler.

The type of language obtained has an important property namely, it can be devided in several levels of complexity. Each level together with the underlying levels constitutes a complete graphics language. This is true in particular for the lowest level, the so-called primitive layer. This layer provides a simple, efficient and easy to learn language for modest graphics applications.

3.5.2 Utilities and other modules

In the application module a library organization must be provided for complete application programs as well as procedures that can be used by any other program. This module also should contain all utility functions for the graphics system that are programmed in the high-level language (like hidden line algorithms).

A special part of this library that will be used by all application programs, contains the graphics I/O routines that perform conversion to and from IL.

There will also exist an administrative module with its own links to the (picture) file system that will enable the user to reserve drawing-machines before an interactive session, realize a batch for picture output, et cetera. This module will provide all kinds of information on a question and answer basis. It can be entered from a normal (nongraphics) terminal.

3.6 The machine driving module

In this module the semantics of the IL for each drawing-machine are defined, through conversion routines. As has been pointed out in the paragraph on languages, this can be a multi-level conversion.

The module also has a link with the file system to store files in low-level machine language format. This link is for exceptional cases only. These files are accessible only for output directly to the drawing-machine.

The basic module can be split up in the obvious way into submodules, one for each machine. In each of the machine modules all real time problems caused by the physical devices must be solved. Also the error recovery from hardware errors or user operating errors must be provided. This might require an extra level of interaction.

4. IMPLEMENTATION

This chapter provides a link between the previous two. We will outline the implementation of the graphics system (chapter 3) on the available hardware (chapter 2). Details will be given only when they are relevant for the purpose of this chapter and when they are available (important information, for instance, about the UNIX system is still lackin9.

It seems appropriate to make a few remarks on our design philosophy and attitude towards system programming. In the first place we have decided to use a high-level system programming language for most programming tasks and to avoid using assembly language as much as possible. Programs which are easier to read and to debug can be expected from this approach. In the second place a thorough study of system programming techniques will be made. In particular we will pay close attention to the ideas developed by P. BRINCH HANSEN [1] (realized in the language CONCURRENT PASCAL). In 4.2 we will show how this approach influences the PDP11/45 operating system.

The discussion in this chapter is split into two parts. The first covers the usage of the SARA configuration, the second that of the PDP11-PDP8 configuration.

4.1 The usage of the SARA configuration

4.1.1 The Mode II protocol

The Mode II protocol will handle the communication between the CYBER installation of SARA and the PDP11/45 satellite. This protocol is designed for the service of a full duplex, asynchronous communication line and has extensive error recovery facilities (see [8]). Up to sixteen independent data streams are supported: eight input streams (to

the host computer) and eight output streams (to the satellite).

This protocol is chosen because it is now supported and will continue to be supported by SARA. The SARA front end system will handle the Mode II protocol irrespective of changes in or replacement of the CYBER installation (see [28]).

The Mode II protocol used by SARA at this moment services the communication between the CYBER and remote batch terminals. Seven of the sixteen streams can be used in this application, namely:

- Two terminal control streams used for communication between the CYBER and the terminal on device level.
- A display and keyboard stream which are the means of communication between the terminal operator and the central site operating system.
- A line printer, card reader and card punch stream.

We intend to use these streams as follows:

- The terminal control streams are used in the obvious manner.

- The card reader and card punch stream are used for the transmission of character or binary information (including pictures) in noninteractive applications.
- The display and keyboard stream will be used for the interactive exchange of information. These streams can -- with some difficulty -- be used for the transmission of binary information too.

In the initial experimental system, we will try to use the display and keyboard streams for the interactive exchange of graphics information. The final system will require assignment of not yet used streams for the following:

- Interactive exchange of binary data.
- Binary output for the HRD-1.

There exists a program for the Mode II protocol that runs under COMTEX [12] on a bare PDP11/20. Large modifications would be required to incorporate it in our operating system on the PDP11/45. The protocol will be written in a high-level system programming language as much as possible. It has been agreed that, during the implementation phase, SARA will offer special testing facilities and manpower.

4.1.2 High-level language and computation speed

One reason for using the SARA installation is the availability of high-level languages. First of all, a graphics language as an extension of ALGOL 68 will be designed. This extension contains utilities such as basic graphics I/O routines and transformations.

Another useful aspect of the CYBER is its high computation speed. In general, all graphics operations that are too time consuming to be executed on the PDP11/45 or are defined on ALGOL 68 data structures, will be delegated to the SARA installation. Complicated graphics utilities such as hidden line and rotation algorithms, projections and specialized I/O routines (typesetting of tables for instance) will be collected in a separate subroutine library, maintained on the SARA installation. These subroutines must be callable both from a high-level language program and interactively from the graphics terminal. They will be written in ALGOL 68 or partly in FORTRAN IV. The latter is chosen for reasons of speed. In any case, as little assembly language (COMPASS [6]) as possible will be used in order to decrease the dependence on the host computer.

4.1.3 SARA background storage and file system

The SARA mass storage facilities will be used for all storage tasks that exceed the capacity of the PDP11-PDP8 disk units. Without modification, the available file system will be used for the following applications:

- To store picture libraries and Digiset character libraries on magnetic tape. These libraries can be part of the graphics system or be user generated.
- The user controlled storage of pictures in the form of files containing IL programs or drawing-machine code.
- The system controlled storage of drawing-machine code files (for example: plot files). When picture output files are too large to be stored on the PDP11-PDP8 disk units, they will be buffered on the SARA disks, and sent to the terminal in parts.

Furthermore, as long as there is no direct transmission line between the Digiset and the PDP11/45, the SARA magnetic tape facilities will be used for the production of Digiset machine code tapes, which can be used as input to the Digiset.

4.1.4 Availability of the graphics system to SARA users

Noninteractive use of the graphics system will proceed as follows: first of all the user generates his picture as an IL file (the output of a high-level graphics language program, for instance). At this stage, SARA resident picture and subroutine libraries can be used. A picture thus generated can be displayed on a graphics device by means of a special "batch" command (see [5]). The identification procedure for output received from SARA is still under consideration.

Interactive work at the terminal can be done in the following way: if necessary, a picture file can be prepared beforehand as in the noninteractive case, and libraries needed during interaction can be retrieved from magnetic tape. Then reservation of graphics devices must be made from a SARA terminal. For the work at the graphics terminal, both the interactive graphics facilities and the standard INTERCOM [3] facilities will be made available. An identification mechanism will be designed that allows the system and the user to keep track of (user) files that may reside at SARA, at the terminal, or both in the form of two copies.

Furthermore, graphics devices must be guarded against damage caused by user programs. The strategy will be to prevent user access to device functions directly and to provide virtual functions instead. These functions can be mapped under system control to device functions in a secure way.

4.2 The terminal configuration

This section is devoted to the function of the PDP11-PDP8 configuration in the IGS. As outlined in the introduction of this chapter, an advanced system programming language will be used for the development of system software. Moreover we intend to implement the IGS as an extension to an existing operating system, if we can find one which meets our requirements.

At this moment the UNIX time-sharing system [24] seems most suited for our purpose for the following reasons:

- It supports an efficient high-level system programming language, named C [25].

- It is designed to allow extensions.
- System development and maintainance is possible under UNIX itself.
- File handling is very flexible.
- It is a time-sharing system.

The UNIX system requires 21K words (42K bytes) of core memory, a disk unit and a clock. The major languages available under UNIX are assembler, FORTRAN IV, and C. The greater part of UNIX software is written in C.

An escape might be the RSX-11M system of DEC [9]. This realtime operating system requires 16K words of core, a disk unit and a clock. Unfortunately FORTRAN IV and MACRO-11 (assembler) are the only languages supported. Here are some other points which make this system less attractive for us:

- No time-sharing facility.
- RSX-11M is written in assembly language.
- More difficult to extend than UNIX.

In the following subsections we will examine several implementation aspects of the graphics satellite system.

4.2.1 Conversion between IL and drawing-machine code programs

The software needed for communication between IL programs and graphics devices will now be considered more closely. This communication proceeds with the help of conversion routines, that translate IL programs to drawing-machine code and, for input devices, from drawing-macine code to IL programs. In some cases, conversion must be separated in two steps: one between an IL program and "extended device" code and one between "extended device" code and physical device code. In the initial system all conversion tasks will reside in the PDP11/45.

In a later stage, the demands made upon the SARA installation and the PDP11/45 system may require reassignment of the work load: permanent delegation of conversions for noninteractive applications to the CYBER may be advisable. In any case, it seems wise to run conversion routines for interactive devices on the PDP11/45 to allow simple local interactive operations such as picture editing.

Graphical devices and their related conversion routines are now treated one by one.

4.2.1.1 The GT40

The GT40 display processor lacks a subroutine and stack mechanism, which would be highly desirable for the realization of display procedures. Software that runs on the PDP11/05 (which is a part of the GT40) will extend the GT40 instruction set with these operations. This will result in core saving (reduced length of display files) and increased interaction possibilities (structured display files). Apart from this addition to the GT40, routines are required for conversion between IL programs and programs in (extended) GT40 code. These conversion routines must compress too large pictures by omitting details. In this way, output for devices like the HRD-1 can be judged on the GT40 first.

4.2.1.2 The HRD-1

The interactive mode of the HRD-1 is superimposed on its storage mode. All interactively produced extensions of a background picture are stored temporarily in a display file. This display file is added to the refresh mode. Upon a user command, a completely new background picture is generated with all extensions included. Deletions can be handled in a similar way. Deleted parts, however, remain visible until a new background is generated. This regeneration clears the refresh mode file. Because the background picture is an IL program, routines are needed to

modify the IL program in accordance with the extensions and deletions made since the previous modification.

The interaction mechanism for the HRD-1 has to be designed from scratch. Because the HRD-1 is developed recently, interactive software is not available.

Like the GT40, the HRD-1 does not have a subroutine mechanism. A solution for this problem might be to extend the HRD-1 instruction set by software on the PDP11/45 in a similar way as proposed for the GT40. Another solution is the use of the system programming language of the PDP11/45 operating system. This language has a procedure mechanism; procedures in assembly language that correspond to HRD-1 operations (hereafter referred to as HRD procedures) can be provided in the form of a library. An IL program can be translated to a system programming language program in a straightforward manner: drawing operations are translated into calls of HRD procedures and IL procedure calls and declarations are translated into the corresponding construction in the system programming language.

Both when an extended HRD is implemented and when translation to the system programming language is used, the basic software provided by Laser Scan Ltd. (which is not interactive) might be of help. Because this software was not developed under a realtime operating system (DOS [13]), large modifications will be required.

4.2.1.3 The plotter

For the plotter, conversion routines are needed that generate a plot file from an IL program. Initially, these plot files will have a structure that is acceptable to the existing PDP8/I plot system. In a later stage, the addition of a subroutine (or macro) facility to this plot system could be considered.

4.2.1.4 The Digiset

The translation of an IL program to a Digiset program is a complicated matter because of the fact that the Digiset machine language is strictly oriented to typesetting. The completely specialized architecture makes the Digiset an ideal testcase for the machine independency of our intermediate language. To allow efficient conversion, alphabets of small straight and curved lines have been, or must be designed (see H.NOOT and P.J.W.TEN HAGEN [23]). The conversion routine must be able to:

- Approximate lines and curves with chaines of basic elements.
- Access libraries of (user defined) special characters.
- Construct typefounts from sets of special characters.
- Split drawings into strips.
- Initialize the Digiset by loading selected typefounts.
- Generate Digiset machine code on the basis of the actions mentioned above.

The study reported in [23] will guide the implementation of most of these tasks.

4.2.2 Local high-level programming language

Especially during the design and testing phase, it is important to have a high-level graphics language that runs on the PDP11/45. We will create such a language by defining a graphics extension of the system programming language using ML/I macros (see P.J.BROWN [2]). Picture definitions in this extended language are preprocessed by the ML/I macroprocessor prior to translation. ML/I macros will be written that replace language elements of the extension by either assembly code or language elements of the host languages. In this way, a graphics language can be realized.

4.2.3 Picture files

There is no difference between a picture file and any other file. From a logical point of view, however, files can be distinguished according to their function in the graphics system. Moreover a specific conversion routine can operate only on certain kinds of files. Within the framework of the existing file system, the following file types will be distinguished:

- High-level language program files.
- Intermediate Language (IL) files.
- Machine code files for specific devices.
- Scratch files for edit operations.
- IL library files.

Routines to perform library search operations will have to be designed and implemented for the handling of IL library files.

5. IMPLEMENTATION STAGES

5.1 Introduction

In this chapter we will outline the order in which the various parts of the IGS will be implemented. The graphics system is designed in such a way that it can be viewed as a structure of independent modules. Each module corresponds with a specific representation or processing facility of graphics data.

As a consequence of this design, modules can be ordered in such a way that the system can be developed in steps by implementing the modules (or part thereof) one after another. The implementation phase is split up into six stages. In each stage several modules are added to the system. After the first stage, a graphics system exists, in which results of high-level graphics programs can be displayed on the GT40. After the second, output to all graphics devices, except the Digiset, is possible. Thereafter, a local picture editor is implemented. Graphical input to high-level programs will be considered in the fourth stage. In the fifth stage, a truly interactive system will arise and finally the Digiset will be tackled. Prior to each stage a detailed plan together with time estimates will be produced. In this way, experience gained during one stage can affect the next. In general the end of each implementation stage seems a natural time for evaluation and documentation. Note: The time needed for evaluation and documentation is not included in this plan.

In section 5.2, we will describe the various system modules in the order in which they are dealt with. For some tasks only the starting time can be indicated. They will extend throughout the implementation phase.

In the following sections we will discuss the implementation stages mentioned above.

5.2 Modules

The programming efforts required are now estimated for each module separately. Estimates related to the first stage are considered more or less accurate, the others only reflect our feeling about the efforts needed.

5.2.1 UNIX (4.2)

The initial operating system UNIX has to be generated. All modules of the first stage must be built and tested under it. We have to study UNIX thoroughly in order to get fully acquainted with it and judge its usefulness. Estimated time: one month.

5.2.2 PP protocol (2.2.2)

The PP protocol will be implemented at this stage in order to make the I/O devices of the PDP8/I configuration available. The work will consist of writing a DR11 driver in assembly language and coding the remainder in C. Estimated implementation time: one month.

5.2.3 Intermediate Language (IL) (3.4)

Because of the central role of the IL in the graphics system, its implementation must be started as soon as possible. Those parts of the IL that are required for control of drawing-machines for output are realized in the first stage.

The interaction parts will be designed later (see stage 4 and 5). Estimated time: output part: one month; completing the IL: three months.

5.2.4 GT40 (4.2.1.1)

The GT40 is the most easily programmable graphics device we have and is needed to experiment with the IL just designed.

Experiments will be carried out with a local high-level programming language (c.f. 4.2.2). This language will be stepwise developed as needed. It will be used in this module for the first time. The time required to develop these language elements is spread over all modules involved.

The programming tasks are:

- Design and implementation of a GT40 - PDP11/45 protocol.
- Implementation of an extended GT40 (4.2.1.1).
- Coding of the conversion routine from IL to extended GT40 code (first stage: output only).

Estimated programming time: three months, interaction part: two months.

5.2.5 Mode II protocol (4.1.1)

The Mode II protocol now has to be implemented. It is a prerequisite for the communication with SARA needed for the implementation of the ALGOL 68 graphics extension.

The programming tasks are:

- The design and programming (in assembly language) of a DQ11 driver.
- The program for the scheduler in C.
- The programming of conversion routines.

Estimated time: three months.

5.2.6 ALGOL 68G (3.5)

With the modules present up to now the implementation of a high-level graphics language will enable testing of one complete output line. The extension of ALGOL 68 to a graphics language will require study of ALGOL 68, primitives of the graphics language, data structures and the possibilities for interaction. First picture output (in the form of IL files) will be considered only.

Estimated time: two months in the initial stage and six for the complete language.

5.2.7 System programming language (4.2)

We have to design and study system programming primitives; this mainly consists of a study of ALEPH [16] and the proposals of BRINCH HANSEN [1]. As mentioned in the introduction of this chapter, this will be done together with others in parallel to the tasks of stage one. Estimated time: two months.

With this module stage one of the system is completed.

OUTPUT TO ONE GRAPHICS DEVICE IS NOW POSSIBLE.

5.2.8 IL file manager (4.2.3)

Before more graphics devices are incorporated in the system, the operating file system has to be equipped with the facility to direct IL files to various devices. When interaction is considered the reverse transports must be possible. The efforts required will strongly depend on the properties of the available file system. In the future the IL file manager will be subordinate to the Interaction Monitor. Estimated time: one month.

5.2.9 HRD-1 (4.2.1.2)

Now the time has come to incorporate other graphics devices, first as output devices only. There are two reasons for this:

- The device independency of the IL must be tested.
- The HRD-1, our most important graphics device, demands a lot of programming effort that has to be started soon.

The work consists of:

- Programming of a HRD-1 driver in assembly language.
- Extension of the HRD-1 with a subroutine mechanism.
- Implementation of a conversion routine from IL to extended HRD-1 code.

At this point the HRD-1 is available as general purpose output device.

Estimated programming time: three months. At a later stage interaction will be considered. For a description of interaction possibilities with the HRD-1 see [19]. The work on interaction with the HRD-1 will probably require two months.

5.2.10 Plotter (4.2.1.3)

With the incorporation of the plotter in the IGS all graphics devices (except for the Digiset) are included in the graphics terminal. Because of the small programming effort needed, it seems natural to do this before interactive work is started. The only program needed is a converter for some IL code to PDP8/I plot-system code.

Estimated time: half a month. At a later stage we might consider the extension of the PDP8/I plot-system with a subroutine or macro mechanis This extension will probably require one month.

OUTPUT TO ALL GRAPHICS DEVICES IS NOW POSSIBLE.

5.2.11 SCOPE (4.1.3, 4.1.4)

So far, the present features of SCOPE have met the requirements for our IGS. A prerequisite for interaction and graphics work is the opening of special (interactive) streams in the Mode II protocol. So, SCOPE has to be modified. We have to specify the streams needed for our graphics system. The adjustments have to be done by SARA. Estimated time: one week (for the specifications only).

5.2.12 Picture files (4.2.3)

Before work on interaction is started, flexible picture file manipulation must be made possible. This requires the implementation of a file system that performs read/write and search operations and has the possibility of distinguishing files of different types. In the future these operations will be controlled by the Interaction Monitor. Estimated time: one month.

5.2.13 Editor (4.2.3)

Graphic output is already possible. Conditions for starting research on implementation methods for interaction are now fulfilled. The design of an editor than can modify local picture files is the first step. Edit commands have to be designed. Estimated time: four months.

LOCAL PICTURE EDITING IS NOW POSSIBLE.

5.2.14 Interaction Monitor and Interaction

After the design and implementation of the local editor, it is possible to experiment with programs and interaction. Embedded in the operating system is the Interaction and Transport Monitor (ITM) that performs the following independent tasks:

- Assignment of (interaction) devices.
- Activating the picture file system and IL manager to perform file operations.
- Interpreting sequences of interaction commands and activating the corresponding functions or returning error messages. For example: the decision whether an interactive command is executed at SARA or at the terminal configuration is taken by the ITM.

The ITM will be implemented in parallel with the study of interaction. First, input of pictures to programs will be dealt with. This requires:

- The implementation of provisional conversion routines between IL and HL.
- The implementation of the first two tasks of the ITM.

Thereafter, the system will be extended to a truly interactive one, which requires the completion of the ITM and the implementation of interaction routines at SARA and at the terminal.

We estimate the time needed up to picture input as three months. The extension to a truly interactive system will require another three months.

5.2.15 I/O Intermediate Language

Up to now simple interaction is possible. All basic tools for an extended interactive use of the graphics devices are available. With the results of the study during the implementation of the ITM, a final definition of the IL can be given, and its internal representation can be fully specified. Next the conversion routines between IL and HL, respectively IL and DM ,called I/O IL, can be implemented in their final form. To avoid machine dependency these routines will be programmed as much as possible in a high-level language.

Estimated time: three months.

INTERACTION ON THE APPLICATION LEVEL IS NOW POSSIBLE.

5.2.16 Utilities

The IGS has now almost reached its final state. Only special operations on pictures such as shading and hidden line elimination, remain to be implemented. Routines for these operations will be collected in a utility library. This approach guarantees an easy extensibility of the system. The utilities will be written mainly in ALGOL 68 and perhaps partly in FORTRAN for reasons of speed. The addition of utilities will result in an adjustment of the ITM. The ITM has to decide whether an interactive command can be carried out by a SARA-resident or terminal-resident utility.

Estimated time: one month for library set up. As long as the graphics system is used, the library will be extended with new utilities.

5.2.17 SARA operating system and reservations

The complete graphics system can now be made available to SARA users. This implies that some extensions of the SARA operating system have to be made, namely:

- A graphics batch command to specify a file and the graphics device that file has to be dumped on.
- A mechanism to relate SARA job identifications to those of picture files.

- An identification mechanism for picture files that are changed at SARA upon commands issued at the terminal by different users. This mechanism has to be part of the facilities offered by interactive streams (5.2.11) shared by several users.
- Accounting procedures for simultaneous use of the terminal and SARA configurations. Most of this work can only be specified by us, but has to be carried out at SARA by others.

Furthermore, a reservation procedure has to be designed, accessible from SARA terminals, by which the usage of graphics devices can be claimed.

Estimated time: two months.

NOW ALL FACILITIES OF THE IGS CAN BE PROVIDED TO USERS VIA SARA

5.2.18 the Digiset

When a satisfactorily functioning graphics system is available, we intend to continue the research reported in [23]. The main use of the Digiset will be for the production of text output. Character sets have to be designed for special texts, like formulas. Producing drawings will be considered too. Furthermore, a conversion routine from IL to Digiset machine code will be implemented (4.2.1.4).

Estimated time: four months.

5.3 Stages revisited

5.3.1 Stage one

This stage, which will end in the possibility to direct the output of high-level graphics programs to the display of the GT40, will start with the study of the initial operating system UNIX (5.2.1),followed by the implementation of the PP protocol (5.2.2). The Intermediate Language

(5.2.3) and the driver and conversions routines of the GT40 (5.2.4) can be designed and implemented. Before the extension of ALGOL 68 (5.2.6) to a graphics language can be implemented, an implementation of the Mode II protocol (5.2.5) has to be made. The goal of this stage is then reached. During stage one a study of system programming primitives is made, which may result in the implementation of a system programming language, suited for our purpose.

The total time for stage one will be one year.

5.3.2 Stage two

To direct output to other graphics devices of the configuration (see fig 2.1), an Intermediate Language file manager has to be implemented. Converters for the various devices such as the HRD-1 (5.2.9) and the plotter (5.2.10) must be made. In the beginning of 1976 a start has to be made with the modification of SCOPE (5.2.11).

The total time of this stage will be approximately five months.

5.3.3 Stage three

In this stage, the work on interaction is started. When picture file manipulation (5.2.12) is possible, we start at a low level with the local editor (5.2.13).

Estimated time: nine months.

5.3.4 Stage four

Interaction with application programs will be realized. We start with an implementation of the ITM (5.2.14), followed by the final definition of the IL (5.2.15). All work on interaction of previous modules will be completed.

Estimated time for this stage: sixteen months.

5.3.5 Stage Five

At this stage the complete interactive system becomes available for SARA users through a reservation system (5.2.17). Furthermore the system will be extended with a library for graphics utilities (5.2.16).

The estimated time: three months.

5.3.6 Stage six

The Digiset will be incorporated in the system (5.2.18).

Estimated time is four months.

MODULE	TOTAL	S1	S2	S3	S4	S5	S6
UNIX	1	1					
PP	1	1					
IL	4	1		2	1		
GT40	5	3		1	1		
MODE II	3	3					
A68G	6	2			4		
SPL	2	2					
IL FILES	1		1				
HRD	5		3	1	1		
PLOTTER	1		1				
SCOPE	1/4			1/4			
PICT FILES	1			1			
EDITOR	4			4			
ITM	6				6		
I/O IL	3				3		
UTILITIES	1					1	
SARA	2					2	
DIGISET	4						4
TOTAL:	50 1/4	13	5	9 1/4	16	3	4

fig 5.2 time schedule

6. LITERATURE

[1] BRINCH HANSEN, P., Operating System Principles, Prentice Hall, Inc, Englewood Cliffs, New Jersey, 1973

[2] BROWN, P.J., ML/I User's Manual, Computing Laboratory, University of Kent, march 1973

[3] CONTROL DATA CORPORATION, Control Data 731-10/173-10 Remote Batch Terminal, Operating and Programming Guide, june 1974

[4] CONTROL DATA CORPORATION, SCOPE 3.2.1, Reference Manual, 1973

[5] CONTROL DATA CORPORATION, INTERCOM 3, Reference Manual, 1973

[6] CONTROL DATA CORPORATION, COMPASS, Reference Manual, 1973

[7] DEHNERT, E., ERNST, G., WETZEL, H., GRAPHEX 68, Graphical Language Features in ALGOL 68, Preprint, Technische Universität Berlin, august 1974

[8] DIGITAL EQUIPMENT CORPORATION, The DQ 11 NPR, Synchronous Line Interface, preliminary, feb. 1974

[9] DIGITAL EQUIPMENT CORPORATION, Introduction to RSX-11M, DEC-11-OMIEA-A-D, may 1974

[10] DIGITAL EQUIPMENT CORPORATION, The PDP 11/45 processor handbook, 1974-1975

[11] DIGITAL EQUIPMENT CORPORATION, The PDP 11 peripherals handbook, 1973

[12] DIGITAL EQUIPMENT CORPORATION, COMTEX - 11 Mod II ISR/TAP, 1973

[13] DIGITAL EQUIPMENT CORPORATION, Disk operating System Monitor programmers Handbook PDP 11, oct. 1972

[14] DR - ING. RUDOLF HELL GMBH, Digiset 40T1/40T2, Befehlsliste und Erlaeuterungen

[15] GROOT, D. and PATBERG, J., General Purpose Graphic System, Preliminary Design Specifications, july 1972

[16] GRUNE, D., BOSCH, R. and MEERTENS, L.G.L.T., ALEPH Manual, Mathematical Centre Report IW 17/74, june 1974

[17] GUEDJ, R.A., The Challenge of Computer Graphics in Continental Western Europe, Proc. IEEE, april 1974

[18] HAGEN, T., The PP-protocol, Mathematical Centre Memorandum, oct. 1974

[19] LASER SCAN LTD., The Laser Scan HRD-1 Laser Display/Plotter Reference Manual, oct 1974

[20] NAKE, F. and ROSENFELD, A., eds, Are we anywhere near a universal graphical language?, Panel discussion in Graphic Languages, Proc. IFIP Working Conference on Graphic Languages, North-Holland Publishing Cy, 1972

[21] BOULLIER, P., et al., METAVISU, in [20]

[22] NEWMAN, W.M., Display Procedures, CACM 14 1971

[23] NOOT, H. and TEN HAGEN, P.J.W., A Digiset Simulator, Mathematical Centre Report IW 30/75, 1975

[24] RITCHIE, D.M. and THOMPSON, K., The UNIX Time-Sharing System, CACM 17(1974) july

[25] RITCHIE, D.M., C Reference Manual, Bell Telephone Laboratory, 1974

[26] RULLY, A.D., A Subroutine Package for FORTRAN, in IBM Systems Journal 7 nos 3 and 4, 1968

[27] SMITH, E.G., The Wiswesser Line-Formula Chemical Notation, McGraw-Hill Book Cy, New York, 1968

[28] STICHTING ACADEMISCH REKENCENTRUM AMSTERDAM, Concept Voorstel tot Aanschaf van een Front-End Systeem voor de CYBER 73-28, SARA B175, march 1974

[29] TEN HAGEN, P.J.W., Grafische Programmeer-talen, Mathematical Centre Syllabus 25, 1974

[30] TEN HAGEN, P.J.W., Het Schrijven van Informatie op een Twee-Dimensionaal Medium, Mathematical Centre Report IN 3/73, july 1973

[31] VAN WIJNGAARDEN, A., ed., Revised Report on the Algorithmic Language ALGOL 68, Technical Report TR 74/3, University of Alberta, Edmonton, Alberta, 1974