ESMERALDA

An Intelligent Satellite for Interactive Graphics

by
Stephen Biering–Sørensen
and
Niels Jørn Nielsen

DAIMI PB–79
June 1977

Institute of Mathematics University of Aarhus
DEPARTMENT OF COMPUTER SCIENCE
Ny Munkegade - 8000 Aarhus C - Denmark
Phone 06 - 12 83 55

Table of contents.

0 Preface.

This paper gives a functional description of *ESMERALDA*, the satellite part of an interactive graphics host–satellite system.

The paper is directed primarily to the reader who

A) wants to know what *ESMERALDA* is, without having to read the program description and the source text

or

B) wants to design a new host–resident part for the system.

ESMERALDA is described in more detail in the authors' masters theses.

Readers of type B will find in these theses the detailed information necessary to implement a new host part. They will also find a description of an already existing host part: the *WIGGIS* system.

We want to thank Geoff Wyvill, University of Bradford, England, for his ideas and assistance during the first phase of the *ESMERALDA*–project.

We also want to thank Peter Møller–Nielsen, University of Aarhus, for good advice during the last year of the project.

1 Introduction.

ESMERALDA is a program which emulates a programmable graphics satellite on DAIMI's interactive graphical terminal (a GT40 with 8K memory). This is connected by a 2400 baud line to the department's DEC–10 computer system, which serves as the host computer for *ESMERALDA*.

ESMERALDA is able to execute actions in accordance with locally stored descriptions, the so–called routines.

The routines are written in an interpreted language. They are transmitted to *ESMERALDA* from an applications program in the host computer. Each routine is stored and can be executed on request. Such a request may come from the applications program in the host, or a running routine, or it may come as the result of the real–time event in the satellite (for example an operator action) with which the routine has been previously associated.

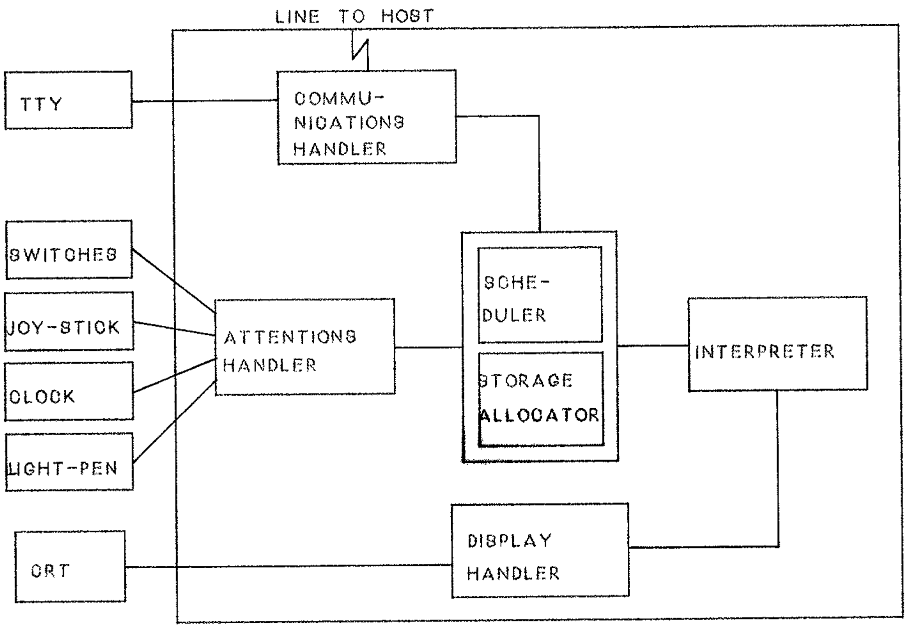
The association of a routine with a specific real–time event is an example from *ESMERALDA*'s repertoire of actions. This repertoire, of course, also includes the creation and manipulation of pictures on the graphical screen.

The transmission of routines from the applications program to the satellite, and the ability to connect the activation of a routine to a real–time event, are the main ideas behind the *ESMERALDA* system.

2 An overview of the system.

In this chapter we describe the various parts of the *ESMERALDA* system.

The following figure shows the *ESMERALDA* machine:



2.1 The interpreter.

The interpreter interprets various descriptions of actions – called *routines*, which have been transmitted to the satellite from the applications program in the host computer.

A routine is executed by inserting a request in a queue, which we will call the ready–queue. Insertion of a request into the ready–queue can be done by a request either from the applications program in the host or from a running routine in the satellite; or it can happen automatically when an event, which the routine has previously been associated with, occurs. For example, a light–pen hit on a menu item can cause the execution of an associated routine.

When a routine is executed it is associated with a so–called data block, a block of storage, where it can store its variables and constants. The association between a routine and its data block takes place when the request for execution is inserted into the ready–queue. This association is completely dynamic. Hence a routine can be executed with different data blocks at different times and a single data block can be associated with several different routines. This will typically be the case when a data block contains some common variables, which many routines use.

2.2 The communications handler.

It is the communications handler's task to handle the communication between the host computer and the satellite.

It either connects the GT40's teletype to the monitor in the host (so that it function like a normal teletype connected to TOPS10) or it connects the satellite (as an entity) to the applications program in the host. The communication takes place in one of two different modes: monitor–mode and system–mode.

In monitor–mode the communications handler works as a simple teletype interface, and the characters are transmitted singly between the teletype and the host.

When the applications program starts executing in the host, the system automatically shifts to system–mode. In system–mode the communication takes place in *records*, that is whole lines.

Records which are written by the operator on the GT40 teletype are always sent directly to the applications program in the host; while records from the host can have four functions. The first of these is

1) Teletype message.
   This is printed directly on the teletype.

In addition there are three types of messages which the applications program uses to control the storage of the routines and the data blocks in the satellite. These are

2) *Build* command.
A record of this type is a description of a routine that is to be stored in the satellite for subsequent execution.

3) *Activate* command.
This command is used to activate an already transmitted routine. It puts a request for the routine's execution into the ready–queue. The data block, which the routine will be associated with upon execution, must also be specified.

4) *Delete* command.
If there is no further use for a routine or a data block stored in the satellite, the host program can request its deletion with this command, thereby releasing the space for other purposes.

The *build* and *delete* commands make it possible for the applications program to control which functions are to reside in the host at any time and which in the satellite. This division of labour is performed at run–time and is completely dynamic.

For instance it is possible for the user to create his own roll–out facilities, if he has a set of routines which he uses only occasionally and therefore does not want to store permanently in the satellite. At the time that he wants to use these routines in the satellite (or preferably some time before, so that the routines have already been transmitted when they are actually needed), he can ask the applications program to transmit them.

It is also possible for the routines in the satellite to communicate with the applications program, because they are able to transmit data blocks back to the host. These data blocks carry with them an identification, which makes it possible for the applications program to discover from which routine the message came.

2.3 The attention handler.

The task of the attention handler is to service those parts of the system, which can give rise to the automatic activation of routines. These parts are: the light–pen, the clock, the panel switches, and the joy–stick.

As far as the first three are concerned, the activation of a routine is caused by a specific event.

An event can be one of:

1) A specific part of the picture on the screen is detected by the light–pen.

2) A specific time is reached.

3) One of the panel switches is changed.

We shall use the expression "to hang" a routine (with an associated data block) on an event. In principle there is a queue for each event. When the event occurs, all the routines in the event queue are transferred automatically to the ready queue. Thus a routine which is added to the event queue will be activated upon occurrence of the event.

As far as the joy–stick is concerned, the attention handling takes place in a slightly different way. There is no event associated with the joy–stick. The routines in the joy–stick–queue are transferred to the ready–queue at every clock–tick, and at the same time the position of the joy–stick is written into the data blocks associated with these routines.

Hanging a routine on the joy–stick is therefore in principle the same thing as hanging it on the clock, delayed by one clock–tick.

2.4 The display handler.

The task of the display handler is to draw a picture on the screen in accordance with a description: a so–called "display file".

This display file is structured in a three–level hierarchy. At the top are the subpictures, which every picture consists of. They are entities with respect to moving pictures on the screen. A subpicture can, for instace, be a whole menu.

Each subpicture consists of a number of atoms. An atom can be a member of only one subpicture. In the case of the menu, the atoms might be the individual menu items: the "light buttons".

An atom is regarded an entity with respect to light–pen detection and changes in light intensity (blink and so on). An atom's starting point on the screen is always relative to the position of the subpicture of which it is a part.

The atoms are composed of lines (vectors) and characters – in the case of the menu, the atom is composed of the single characters in the menu item and perhaps four lines forming a frame around the word.

It is not possible to make selective changes in an atom, but it is possible to erase the whole content of an atom and redraw it in a changed version. This will typically be the case, when the user wants to perform "rubber–banding".

2.5 The scheduler.

The scheduler handles the coordination of the execution of the routines inserted into the ready–queue. When a routine has finished executing, the scheduler takes the next element from the ready–queue, and, in the case that it is a routine, control is transferred to the interpreter, which takes care of the execution of the routine.

However the ready–queue is also used for other purposes than routine execution. The system itself uses it to schedule tasks which must not coincide with the execution of routines, for example releasing of storage, which may cause compression within the store. In the case of such a system task, the scheduler itself takes care that it is performed.

2.6 The storage allocator.

The task of the storage allocator is to administer the storage of the satellite. When space in the store is requested by other parts of the system (for example in the form of data blocks), the storage allocator will allocate this space and hand it over to whoever asked for it. The storage allocator also handles the releasing of space which is no longer needed.

3 Parallelism.

It should be emphasized that, in principle, the communications handler, the attention handler, the interpreter, and the display handler run in parallel (multi–programmed). In fact the display handler really does run in parallel with the others.

Hence it may appear to the operator at the satellite that many tasks are being performed simultaneously. For instance the host can transmit a new routine to the satellite, while routines are being executed which cause pictures to move continously on the screen. And at any time the operator may use the joy–stick or the panel switches as well as the light–pen.

Thus the system may be performing useful work even while it is waiting for action by the operator (e.g. a light–pen hit).

4 Routines and primitives.

An *ESMERALDA* routine is in principle a string of calls to a set of primitives. Each call will normally be followed by a parameter list. A parameter is an address in the data block associated with the routine during its execution.

The primitives currently available are listed below, together with a short description of their function.

4.1 Events and routine activation.

activate.

Parameters: routine, data block.

Function: activates the routine. In other words it adds the routine together with the data block to the ready–queue for subsequent execution.

hangatom.

Parameters: routine, data block, atom.

Function: adds the routine together with the data block to the atom's event–queue, thereby requesting activation of the routine upon the next light–pen detection of the atom. When this light–pen detection occurs, the attention handler will automatically write the (x,y) position, the subpicture, and the atom of the hit into the first four cells of the data block.

hangclock.

Parameters: routine, data block, number of clock–ticks.

Function: adds the routine together with the data block to the event–queue of the N'th clock tick, counted from the time the request is made. (Where N = number of clock ticks)

hangkey.

Parameters: routine, data block, key number, position.

Function: adds the routine together with the data block to the event–queue of the given combination of key (i.e. panel switch) and position (up/down).

hangjoystick.

Parameters: routine, data block.

Function: adds the routine together with the data block to the joy–stick–queue, thereby requesting its activation upon the occurrence of the next clock tick.

unhangatom.

Parameters: routine, atom, reply variable.

Function: if the routine is in the event–queue of the atom, the first occurrence of it is removed and a pointer to the data block of that occurrence is written into the reply variable. Otherwise nothing is done and the reply is *nil*.

unhangclock.

Parameters: routine, reply variable.

Function: if the routine is in the event–queue of any clock tick, the first occurrence of it is removed, and a pointer to the data block is written into the reply variable. Otherwise nothing is done and the reply is *nil*.

unhangkey.

Parameters: routine, key number, position, reply variable.

Function: if the routine is in the event–queue of the given combination of key and position, the first occurrence of it is removed and a pointer to the data block of that occurrence is written into the reply variable. Otherwise nothing is done and the reply is *nil*.

4.2 The display file.

newsubpic.

Parameters: reply variable.

Function: a pointer to a new (empty) subpicture is written into the reply variable.

newatom.

Parameters: size, reply variable.

Function: a pointer to a new (empty) atom is written into the reply variable.

addtosubpic.

Parameters: atom, subpicture.

Function: adds the atom to the subpicture.

removeatom.

Parameters: atom, subpicture.

Function: if the atom belongs to the subpicture it is removed from it. Otherwise nothing is done.

display.

Parameters: subpicture.

Function: adds the subpicture to the display file.

undisplay.

Parameters: subpicture.

Function: if the subpicture is in the display file it is removed. Otherwise nothing is done.

moveto.

Parameters: subpicture, x, y.

Function: moves the subpicture so that its starting point is at (x,y).

moveby.

Parameters: subpicture, x, y.

Function: moves the starting point of the subpicture by the vector (x,y).

transformpic.

Parameters: subpicture, m_{11}, m_{12}, m_{21}, m_{22}.

Function: transforms the subpicture by left multiplying all the vectors of all the atoms of the subpicture by the transformation matrix

$$\begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix}$$

move.

Parameters: atom, x, y.

Function: appends a dark vector (x,y) to the atom.

draw.

Parameters: atom, x, y.

Function: appends a bright vector (x,y) to the atom.

character.

Parameters: atom, character.

Function: appends the character to the atom.

defineatom.

Parameters: atom, attributes.

Function: changes the attributes of the atom in accordance with the parameter. The attributes are light intensity, blink, and line type; and they are coded into a single word.

reduceatom.

Parameters: atom.

Function: reduces the space occupied by the atom to the least possible.

cleanatom.

Parameters: atom.

Function: erases all the vectors and characters of the atom.

4.3 Flow control.

ifequalgoto.

Parameters: a, b, offset.

Function: if $a = b$ a jump is performed to the relative code position specified by the offset (i.e. the next N bytes in the routine are skipped over, where N = offset, $-127 \leqslant N \leqslant 128$). Otherwise nothing is done.

iflessgoto.

Parameters: a, b, offset.

Function: if $a < b$ a jump is performed to the relative code position specified by the offset. Otherwise nothing is done.

gotolabel.

Parameters: offset.

Function: a jump is performed to the relative code position specified by the offset.

test.

Parameters: kind.

Function: if kind = 0 the system is halted. Otherwise the system is put into a waiting state until a rubout is typed on the teletype. This primitive has proved useful as a debugging aid.

exit.

Parameters: none.

Function: terminates the execution of the calling routine and gives the control back to the scheduler. All routines must end with a call to exit.

4.4 Data.

getdatablock.

Parameters: size, reply variable.

Function: a pointer to a new data block is written into the reply variable.

delete.

Parameters: block (either data block, routine, atom, or subpicture).

Function: adds a storage allocator request to the ready–queue, telling it to release the space occupied by the block.

copyto.

Parameters: data block, variable, data.

Function: copies the data to the given variable of the given data block. Data is an address in the current data block.

copyfrom.

Parameters: data block, variable, reply variable.

Function: the content of the given variable of the given data block is written into the reply variable.

plus.

Parameters: a, b, reply variable.

Function: the sum of the variables a and b is written into the reply variable.

minus.

Parameters: a, b, reply variable.

Function: the difference of the variables a and b is written into the reply variable.

datalength.

Parameters: data block, reply variable.

Function: the size of the data block is written into the reply variable.

time.

Parameters: reply address.

Function: the number of clock ticks since start–up is written into the reply variable.

4.5 Communication.

replytohost.

Parameters: identification, data block.

Function: sends the identification and the contents of the data block back to the host.

texttotty.

Parameters: data block.

Function: types the contents of the data block (interpreted as character values) on the teletype.

savedisplayfile.

Parameters: none.

Function: sends a coded description of the entire display file back to the host. This primitive facilitates creation of "hard" copies of a displayed picture.

5 An example.

To illustrate the use of *ESMERALDA* we give the following example. Assume that we have already created a number of subpictures each consisting of a number of atoms. We now want to be able to move these subpictures under the control of the operator.

The way we do this is as follows: the operator selects a subpicture by pointing to one of its atoms with the light–pen. From now on this subpicture can be moved using the joy–stick until another subpicture is selected in the same way. This continues until the button on the top of the joy–stick is pressed.

In the following descriptions the names in quotation marks are symbolic names. If a parameter is written in *italics* it is a constant, otherwise it is a variable in the data block.

To implement the conversation, we need a data block, which might look like this:

```
"DATA":        "HITX"
               "HITY"
               "HITSUBPIC"
               "HITATOM"
               "JOYX"
               "JOYY"
               .
               .
               .
```

and the routines:

"REHANGER": **HANGATOM**
"REHANGER"
"DATA"
"HITATOM"
EXIT

and

```
        "MOVER":          IFEQUALGOTO
                          "JOYBUTTON"
                          1
                          "STOP"
                          MOVEBY
                          "HITSUBPIC"
                          "JOYX"
                          "JOYY"
                          HANGJOYSTICK
                          "MOVER"
                          "DATA"
                 "STOP":  EXIT
```

REHANGER, together with the data block DATA, must be hung onto all the atoms in the subpictures we want to be able to move.

MOVER, together with DATA, must be hung on the joy–stick.

Let us explain in detail what is going on.

The routines REHANGER and MOVER have been transmitted from the applications program in the host, and the data block, DATA, has either been generated by another routine or has been transmitted in the same way as the routines.

When an atom is selected by the light–pen, the routine REHANGER is transferred to the ready–queue, and the names of the atom and the subpicture are written into the first cells of DATA together with the coordinates of the point which was hit. REHANGER's only task is to hang itself back onto the hit atom, because the information necessary for the moving operation, namely the name of the subpicture, has already been written into DATA.

At the next clock–tick, the routine MOVER will automatically be transferred from the joy–stick–queue to the ready–queue, and the position of the joy–stick is written by the system into the data block DATA.

The task of the routine MOVER is as follows. If the joy–stick top button has not been pressed, then the selected subpicture will be moved according to the (x,y)–position of the joy–stick, and the routine will hang itself back onto the joy–stick–queue for subsequent execution upon the next clock–tick.

However if the joy–stick top button has been pressed, then the routine MOVER will do nothing, and consequently not hang itself back onto the joy–stick. This will stop the conversation, because the joy–stick will now be inactive until something else is hung on it.

The routines and the data block are, except for the symbolic form, shown as they will look in *ESMERALDA*. Of course it is not our intention that an applications programmer write his programs this way. There will be an interface in the host computer to "translate" the routines from a more readable form to the form which *ESMERALDA* requires. An interface of this kind called *WIGGIS* is provided, but it is beyond the scope of this paper to go into details of it. Further information can be found in the theses mentioned in the preface.