

A PROPOSAL FOR AN INTERACTIVE ANALYSIS ENVIRONMENT INCLUDING GRAPHICS

The First Approach

This note shows the basic elements of a framework for interactive analysis and graphics. It is an approach to a first implementation of the ideas that were discussed in the group on graphics requirements [1]. Even though our intention is to provide all the required facilities, they do not all appear in this paper.

We are presently testing the feasibility of our proposal by writing a prototype. Some thinking still has to be done, and not all the elements given below are described in their final form. Nevertheless we considered that it was necessary to talk about our ideas now in order to open the discussion. Critical remarks and additional propositions are welcome.

The ideal device would be a workstation, giving flexibility and ease of implementation by its windowing and multiprocessing facilities. But we are aware of the fact that the analysis will be done on a very large range of devices, including low level 2D terminals. The proposed architecture is suited to either environment, though performance will inevitably be superior on a workstation. On a workstation we will take advantage of the native capabilities whereas on a simple terminal we must write additional software to provide similar facilities.

Menus, though a friendly way of presenting choices to the user, are not discussed here. We restrict ourselves, for the purpose of this note, to the basic elements of communication.

The proposed framework relies on the data model adopted in Aleph. The different modules shown have in common a unique data structure formulated according to this model. To gain maximum benefit from the regular data structure, the data modelling should be complete and consistent. Ad-hoc solutions may be provided if required, but bring with them a loss of flexibility. For those who wish, a simple FORTRAN interface via common blocks will be provided, this introduces of course much more rigidity.

Comments to the structure chart:

Overall architecture

The architecture is essentially flat, a very simple steering interpreter calling many different modules. The steering interpreter does nothing else than interpret commands, coming directly or indirectly from the user, by a call to the corresponding module.

A structuring is introduced dynamically : the user will normally address the system by a compound command (macro) triggering a complete set of basic actions. These macros are interpreted and one basic command after the other is sent to the steering interpreter to be executed.

Command style

A command is of the form

VERB/QUALIFIERS ERE1, ..., EREi, ...

following the well-known syntax of the Vax DCL. It consists of 3 parts:

- a verb, describing the action to be taken and being in correspondance either to a basic module or to a macro
- qualifiers specifying parameters for the command or macro
- objects in the data structure that are addressed in the form of Entity Relationship Expressions (ERE).

We should like to have exactly the same syntax in case the verb specifies a compound action and not a basic one. We are presently studying the problem of propagating the qualifiers and/or objects to the basic actions.

Program modules

The program modules that are represented in the structure chart are of different types:

1) standard modules, used to perform service functions like "get event" and "histo viewer". They correspond to basic commands.

2) graphics modules for the display of objects, for identifying graphical quantities or changing the aspect of a display (draw, pick, viewing server).

3) user or analysis modules, shown by "Pi". They may be modules taken from the reconstruction program (like fit of a track), or analysis modules, or anything the user would like to integrate in this framework. In order to make this integration possible, a minimum number of basic rules have to be observed. A module, not primarily written for this framework, can be integrated by adding a small dialogue part which verifies the presence of the necessary information and possibly interacts with the user to get it (see appendix 2). The constraints for a module to be integrated in this framework will be formulated in a later paper. All of them may call the basic service packages (dialogue handler, EARL, window manager and histogramming package) described below.

4) service packages. They consist of a set of routines that can be called from any of the basic modules (user or standard ones). Their objectives are to get and digest the commands of the user (dialogue handler), to address objects in the data structure (EARL), do histogramming (Histo package), create windows and communicate with them (window manager). " Create Macros" and "Line IO" are secondary to these service packages.

service packages:

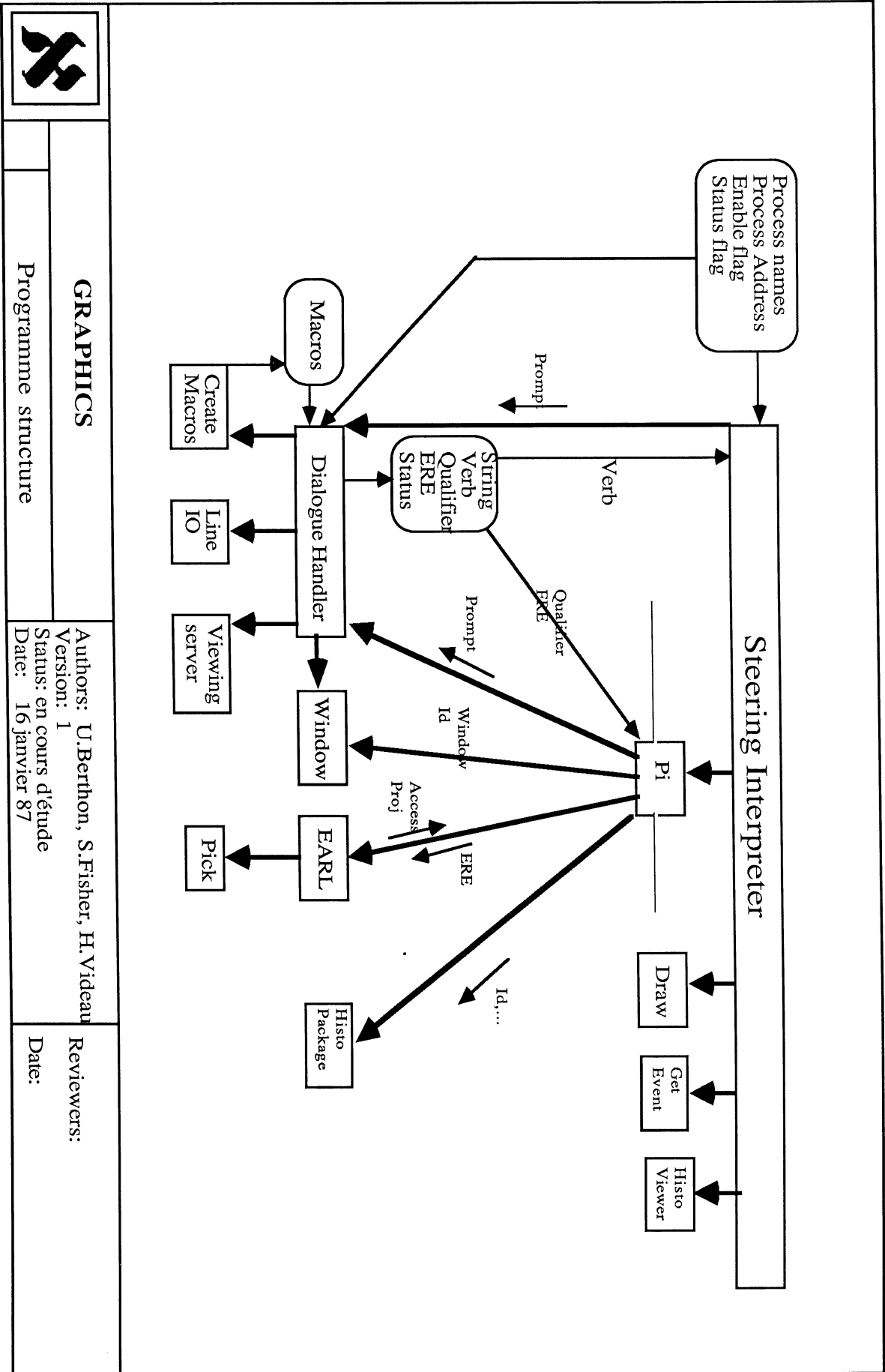
- the dialogue handler gets the user command, of the form given above, for the moment not by a menu but by simple command line input. It splits it up into its 3 parts (verb, qualifiers and EREi), and stores it together with these 3 parts such that the module addressed by the verb can get it. It knows the macro definitions and , in case the verb is a macro, cracks it up into its components of verbs addressing directly the basic modules, and delivers only one basic verb at a time. It handles as well the interpretation of the qualifiers and translates them into a table of parameters delivered to the module concerned. Since it knows the basic verbs, it can distinguish whether the command was addressing the specific module or asking for activation of one of the other basic modules. This is signalled by a special status flag, and the module has in this case to give control back to the steering.
- the window manager may reserve a window on the screen for graphics and for text I/O, such that each process can have its I/O in one or several separate windows. These windows may be moved and their size and background may be altered.
- EARL is an interpreter of an entity relationship expression ERE. It allows the user to address objects in the data structure by their relations and/or to formulate conditions (e.g. "all clusters associated to a certain track" or "all tracks with momentum greater than something"). Its output is the access to the object(s) chosen by the ERE.
- the histogram package is presently under study, in collaboration with the online group.

References:


- [1] Graphics Requirements Group (Bowdery, Brandt, Cordier, Drevermann, Fernandez, Knobloch, Mermikides, Rolandi, Schlatter, Videau)
Requirements for a Graphics and Interactive Programming Environment
ALEPH 87-1, NOTE 1

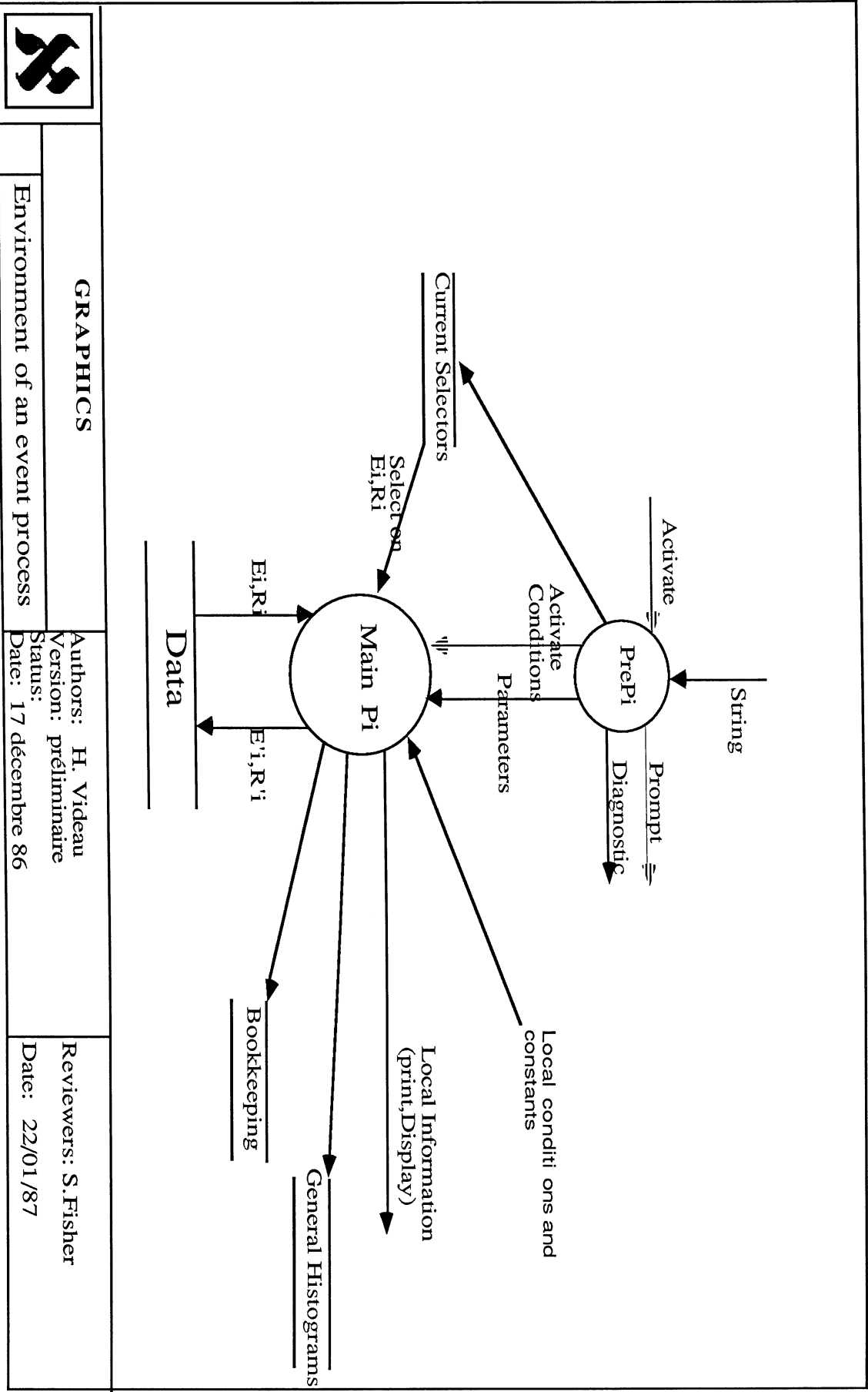
Appendices:

- (1) Structure chart of the entire framework .
- (2) Data flow diagram showing the environment for a process to be integrated into this framework .
- (3) Entity-Relationship diagrams .
- (4) Tentative Data Description.



APPENDIX 1

	GRAPHICS		Auteurs: U.Berthon, S.Fisher, H.Videau Version: 1 Status: en cours d'étude Date: 16 janvier 87	Reviewers: Date:
	Programme structure			



GRAPHICS		Authors: H. Videau Version: préliminaire Status:	Reviewers: S. Fisher Date: 22/01/87
Environment of an event process		Date: 17 décembre 86	

Data flow diagram



APPENDIX 3

Remarks to help understanding the graphics Entity-Relationship diagrams.

CONTEXT

The context shows the distinction between the structure of the data as they appear in the reconstruction programme, the purely graphical structure and the package used to handle the drawing: GKS.

DATA STRUCTURE

The data structure expressed in the diagram does not follow exactly the structure defined for the reconstruction which is still not formalised entirely. This will be corrected in time and what we give here is merely an example. The diagram exhibits a double structure due to the geometry of the detectors and to the physics: for example the TPC coordinates belong to a TPC sector and to a track.

The entity sets, having names ending with a // sign contain information, derived from the main entities and therefore in one to one correspondance with them. This information is needed for the graphical representation of the objects, for example:

TPC// expresses the TPC coordinates in different systems like $x, y, z, \rho, \theta, \dots$

Track// contains the helix parameters plus the starting and stopping points needed to draw the tracks.

GRAPHICS STRUCTURE

The Graphical Object makes the connection between the object (see Data Structure), its Representation, its Aspect and the graphical Segments in which it will be drawn. It has for attribute the level of detail with which you want to draw this object.

The Representation tells you how an object will be drawn. Example: a TPC coordinate can be just a symbol or an error bar or both, a storey can be a dot or have its exact shape. It tells you what primitives are used to draw it.

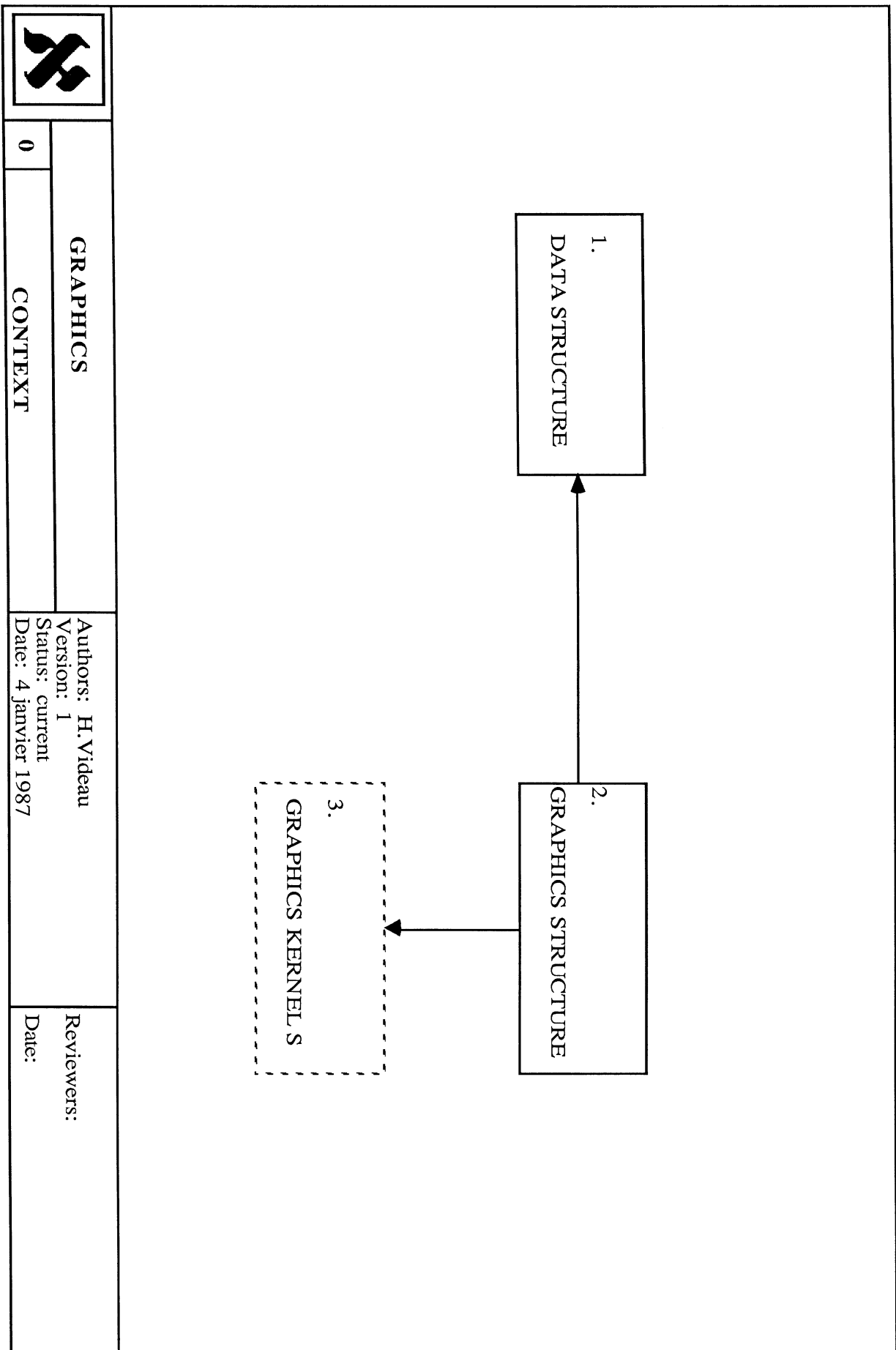
The Aspect is the choice of symbol, colour, etc. It connects to the GKS bundles. The actual aspect is then handled at the level of the workstation.

The Views entity defines what type of view (2D or 3D) and the type of coordinates to plot.

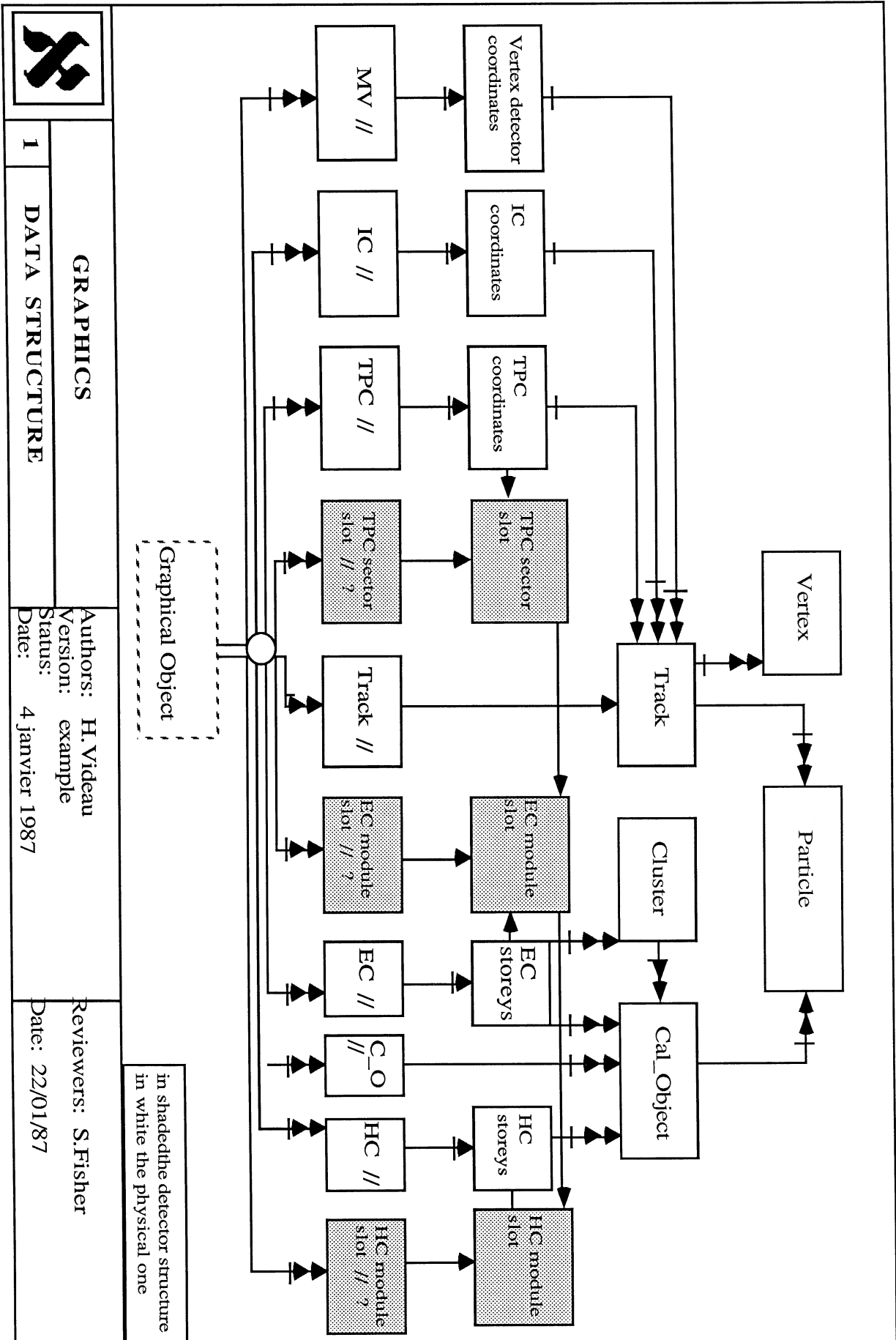
The Logical Segment regroups Graphical Objects to be treated as a whole. It may be represented in more than one view. Its attributes contain the ones for a GKS segment: Id, dimension, visibility, detectability, highlighting, priority. It is associated to transformations which can be performed on segments as a whole.

The Actual Segment is simply the relationship between Views and Logical Segments. More than one Logical Segment can be on a view and a Logical Segment can be on more than one view.

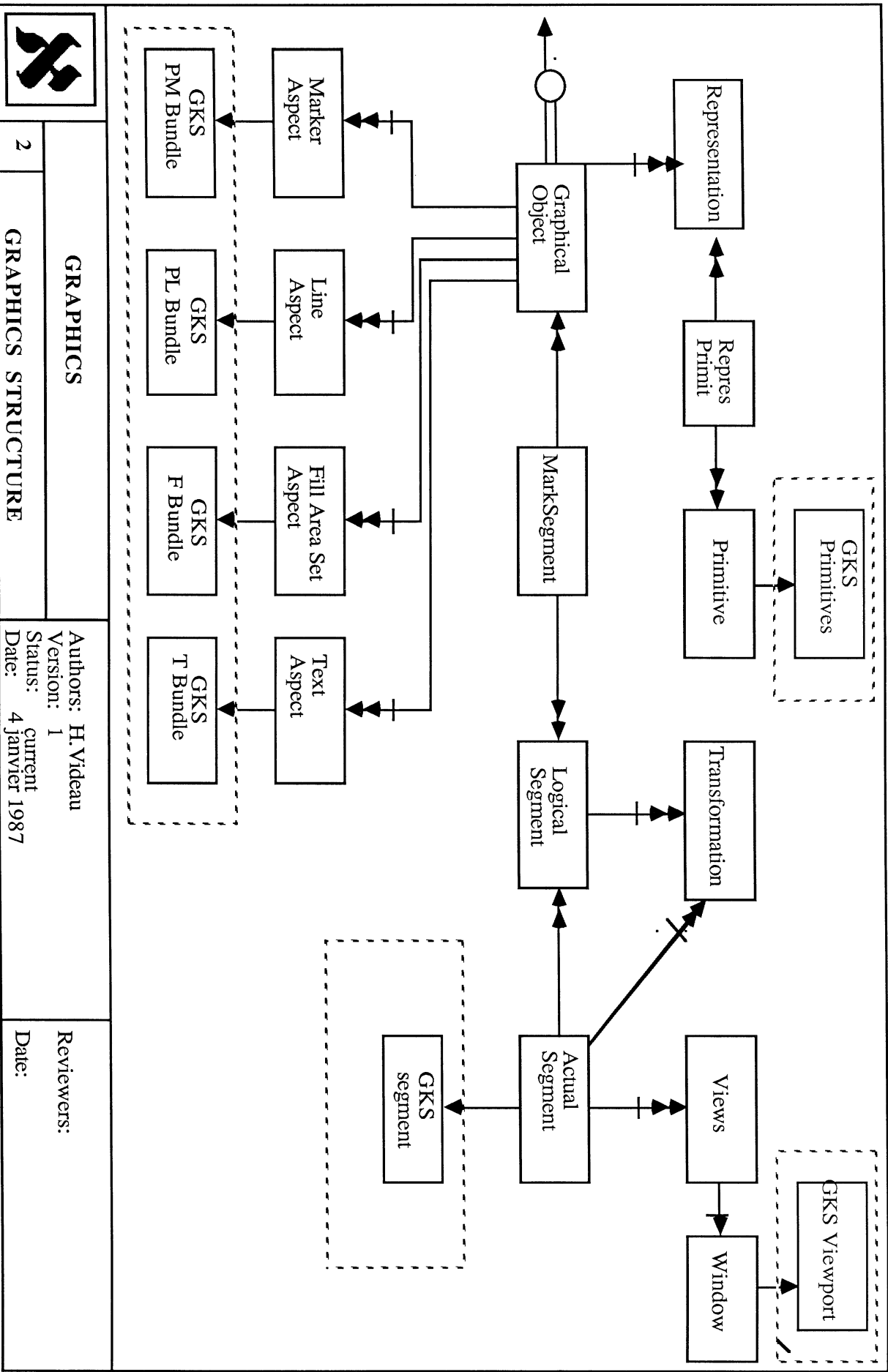
Mark Segment makes the relation between Graphical Object and Logical Segment. Its fundamental attribute is the Pick Identification which is used to select an object by picking its representation on the screen.



Entity-Relationship diagram



Entity-Relationship diagram



Entity-Relationship diagram

APPENDIX 4

Tentative Data Description

SUBSCHEMA Graphics

: 'This exhibits the entity-relationship structure of the graphics'

AUTHOR 'H.Videau'

VERSION '1'

DATE '8 decembre 1986'

DEFINE ATTRIBUTE

Name = CH16
: 'a name of 16 characters maximum'
;

Length = REAL
: ' length in cm'
;

END ATTRIBUTE

DEFINE ESET

GraphicalObject

: 'defines a graphical object as a relation between a physical object,\
a graphical representation, aspect parameters and segments'
=(DetailLevel = INTE [1,4])
;

Representation

: 'Type of graphical representation , ex: symbol,error bars,...\
will be filled when thoroughly studied'
= (Name)
;

RepresPrimit

: 'many to many relationship between Primitives and Representation,\
tells you what primitives are drawn to figure the object'
;

Primitive

: 'This entity maps on the GKS entities '
=(Name)
SIZE 4,4
;

MarkSegment

: 'tells you what GraphicalObject belongs to what LogicalSegment'
=(PickId = INTE [1,*] : ' Pick identification')
;

LogicalSegment

: 'decomposition of the picture in graphical segments not considering\
its appearance in different views, collection of GraphicalObjects'
=(ID = INTE : ' identification',
DI = INTE 2|3 : ' dimension 2 or 3',
VisFlag = LOGI : ' visibility flag',

```

    DetFlag      = LOGI : ' detectability flag',
    HiLight      = LOGI : ' high lighting flag',
    Prior        = INTE : ' priority')
;

```

ActualSegment

```

:'the graphical segments appearing in each particular view,it is in fact\
 an entified relationship between view and LogicalSegment'
=(ExFlag      = LOGI : ' existence flag')
;

```

Views

```

:'the way we look at the objects'
=(ProjDim     = INTE [2,3]: ' dimension of the projection',
  CoordChoice = INTE [1,*]: ' choice of coordinates')
;

```

Windows

```

:'Set of windows on the screen '
=(Posit (3)   = Length : ' Position of the window',
  WinSize (3) = Length : ' Size of the window',
  BackCol     = INTE   : ' Background colour',
  Title       = CH32   : ' Title',
  Type        = INTE   : ' Type',
  Font        = INTE   : ' Font' ,
  CharSize    = INTE   : ' Character size' ,
  CharSpace   = INTE   : ' Character spacing' ,
  ColInd      = INTE   : ' Colour index' ,
  Status      = INTE   : ' Status')
;

```

MarkerAspect

```

:' appearance of a marker associated to a GraphicalObject'
=(MarkType    = INTE [1,4]: ' marker type',
  MarkSize    = INTE      : ' marker size',
  ColInd      = INTE [1,*]: ' colour index')
;

```

LineAspect

```

:'appearance of a line associated to a graphical object'
=(LineType    = INTE [1,4]: ' line type',
  LineSize    = INTE      : ' line size',
  ColInd      = INTE [1,*]: ' colour index')
;

```

FASAspect

```

:'appearance of a fill area set associated to a graphical object'
=(PatType     = INTE [1,4]: ' pattern type',
  ColInd      = INTE [1,*]: ' colour index')
;

```

TextAspect

```

:'appearance of a text associated to a graphical object'
=(TextString  = CH16     : ' text string',
  ColInd      = INTE [1,*]: ' colour index',
  Size        = INTE     : ' size',
  Spacing     = INTE     : ' spacing' ,
  Font        = INTE     : ' Font' )
;

```

END ESET

DEFINE RSET

```
(GraphicalObject [1,1] -> [0,*] Representation)
  : 'a graphical object has a representation'
  ;

(RepresPrimit [1,1] -> [1,*] Representation)
  : 'a primitive may be used in more than one representation'
  ;

(RepresPrimit [1,1] -> [1,*] Primitives)
  : 'a representation may use more than one primitive'
  ;

(MarkSegment [1,1] -> [1,*] GraphicalObject)
  : 'a graphical object may belong to more than one segment'
  ;

(MarkSegment [1,1] -> [1,*] LogicalSegment)
  : 'one segment may contain more than one object'
  ;

(ActualSegment [1,1] -> [1,*] LogicalSegment)
  : 'the same logical segment seen in different views corresponds\
    to different actual segments'
  ;

(ActualSegment [0,1] -> [0,*] Views)
  : 'the same logical segment seen in different views corresponds\
    to different actual segments'
  ;

(Views[1,1] -> [0,1] Windows)
  : 'To a view is associated the Window in which it is represented'
  ;

(GraphicalObject [0,1] -> [1,*] MarkerAspect )
  : 'a graphical object has an aspect for a composing marker '
  ;

(GraphicalObject [0,1] -> [1,*] LineAspect )
  : 'a graphical object has an aspect for a composing line '
  ;

(GraphicalObject [0,1] -> [1,*] FASAspect )
  : 'a graphical object has an aspect for a composing Fill Area Set '
  ;

(GraphicalObject [0,1] -> [1,*] TextAspect)
  : 'a graphical object has an aspect for a composing text '
  ;

(GraphicalObject [0,1]    -> [1,*] Par1|
                          -> [1,*] Par2|
                          -> [1,*] Par3
  BY ObjectType)
  : 'a graphical object is associated to an object in a parallel set'
  ;
```

END RSET

END SUBSCHEMA