GG

29 AVR. 1988

CERN-PRE 88-014

4

RAL-88-024

Science and Engineering Research Council

Rutherford Appleton Laboratory

CHILTON, DIDCOT, OXON, OX11 0QX

RAL-88-024

CERN LIBRARIES, GENEVA

CM-P00052016

Modelling and Building Graphics Systems: GKS

M Boano, R Brazioli, S M Fisher, P Palazzi and W R Zhao

March 1988

PLEASE
MAKE A
PHOTOCOPY
or check out as
NORMAL
LOAN

Modelling and Building Graphics Systems: GKS¹

W. R. Zhao

IHEP Academica Sinica, P.O. Box 918, Beijing, China

M. Boano², R. Brazioli, P. Palazzi

CERN DD Division, CH-1211 Geneva 23, Switzerland

S. M. Fisher

Rutherford Appleton Laboratory, Chilton, Didcot, Oxon OX11 0QX

March 7, 1988

¹submitted to CG International '88

²now at Digital Equipment Corporation, F-06561 Valbonne, France

MODELLING AND BUILDING GRAPHICS SYSTEMS: GKS.

M. Boano, R. Brazilioli, S. M. Fisher, P. Palazzi, W. R. Zhao

ABSTRACT

The GKS-2D standard specifications were expressed by a combination of the Entity-Relationship (ER) model with Dataflow and State Transition diagrams. Our description was not only precise and complete, but also easier to understand than the original standard, mainly because of the use of diagrams. Starting from this description a GKS system was rapidly constructed making use of a general purpose subroutine package for ER data definition and manipulation. The performance of this GKS was found to be similar to that of commercially available products. Our description was extended to GKS-3D and is generally applicable to any graphics system. We suggest that such an approach would lead to more rapid convergence in the design of graphics standards and the construction of the corresponding systems.

INTRODUCTION

The Graphical Kernel System (GKS) is registered as an ISO Standard for computer graphics programming. It is specified in the Standard by words - with a few informal diagrams to aid understanding.

GKS provides a set of low level functions for graphics programming that can be used by applications to produce computer generated pictures. It provides a functional interface between an application program and a configuration of graphical input and output devices at a level of abstraction that hides the peculiarities of device hardware.

The standard took a long time to produce. GKS may be considered to have been started at the Seillac I meeting in May of 1976, and GKS 7.2 was a Draft ISO standard ISO/DIS 7942 in 1983. It became a full standard in 1986. Commercial packages were slow to appear. Tektronix produced a level 2b version in 1984, and Digital a 0b version in 1985.

To shrink this enormous time span, which results in obsolescent products, one must move rapidly from first specification to a working system. For this to be possible the specification must be clear to allow wide exchange of ideas and there must be a simple way to construct the system from its specification.

EXISTING SPECIFICATIONS

GKS <ISO82> is specified in the standard mostly by words and a few diagrams. Figure 9 of the standard, for example, attempts to provide an overview of a GKS system. It looks plausible, but, when examined in detail, it is not possible to check the diagram in any formal way because it uses symbols in an inconsistent manner.

Definitions of objects are widely scattered. For example, the polyline is first mentioned in the GKS standard in the list of definitions near the beginning (page 9):

"polyline: A GKS output primitive consisting of a set of connected lines".

This is not a complete description. The fact that a polyline has attributes (for example LineWidth) is only mentioned much later in section 4.4.3. Objects are generally understood much better when their attributes are identified. Chapter 6 of the standard describes the data objects of GKS but not their inter-relationships.

The Vienna development method (VDM) has been recently applied to GKS specification <DUCE86>. This is an algebraic method, which though precise, takes time to understand. This would reduce the value of the specification at the earliest design stage, where free exchange of ideas is vital. We also know of no simple path from a VDM specification to a working system.

SPECIFICATION WITH ER, DATA FLOW AND STATE TRANSITION DIAGRAMS

Our specifications make use of the Entity-Relationship (ER) model, coupled to the dataflow diagrams (DFD) of Structured Analysis <DEMARCO78> and state transition diagrams. Mathematical transformations were specified using FORTRAN statements.

Entity-Relationship

A data model consists of the definition of data structures, of operations on the data and integrity constraints. The ER model <CHEN76> organises data as entities which are objects characterised by their properties (known as attributes) and their relationship to other entities. An ER diagram is a pictorial representation of sets of entities (denoted by a rectangle) and their relationships (shown by an arrow). The attributes of the entity set are not shown on these diagrams, but are present in the complete description represented by the Data Definition Language (DDL).

The complete GKS-2D is described by 32 Entity Sets, which have been grouped into four ER diagrams. Figure 1 shows one of the four: the 'PICTURE'.

In GKS, pictures are constructed from a number of basic blocks, called output primitives, which are abstractions of the basic actions that a graphical output device can perform (e.g. drawing a line). There are six output primitives in GKS: polyline, polymarker, text, fill area, cell array and generalized drawing primitive (GDP), each of which has associated with it a set of parameters, for a particular instance of the primitive.

GKS allows output primitives to be grouped together into units known as segments which are stored and may be manipulated as a single object.

A segment may contain many graphics primitives; it may also be empty. This is shown by the word 'Segment' inside a rectangle representing the entity set 'Segment'. The relationship between 'Segment' and 'Primitive' is shown by the arrow. The 2 arrow heads indicate that there may be more than one instance of 'Primitive' in a 'Segment', and the bar by the arrow heads shows that there may be none.

A 'Segment' may be drawn many times on a workstation 'Ws', but must always be drawn at least once. Work Station Independent Segment Storage - the WISS is regarded as a workstation. A workstation may exist with no segment or it may have many drawn on it. Each instance of a 'Segment' being drawn on a workstation 'Ws' corresponds to an entity in the entity set 'Draw'.

Entity-Relationship diagram

MacADAMO

Figure 1: The 'PICTURE' ER diagram

DEFINE ESET

```

PolyLine : 'a 2-D graphical point for Polyline primitive'
          = (Index = INTE           : 'Index in a GKS sense',
             LType = INTE [1,4]     : 'linetype : GLSOLI=1
                                     GLDASH=2
                                     GLDOT =3
                                     GLDASD=4',
             LWidth= REAL            : 'width scale factor relative
                                     to WS standard line width',
             LnCol = INTE            : 'individual colour');

```

```

LPoint   : 'a 2-D graphical point for Polyline primitive'
          = (P(2) = REAL            : 'x = P(1), y = P(2)');

```

DEFINE RSET

```

(LPoint [1,1] -> [2,*] PolyLine )
: 'each polyline is related to at least 2 points';

```

Figure 2: An extract from the DDL

Figure 3: Top level GKS data flow diagram

GKS

0

GKS

Author: M. Boano
Version: 1.0
Status: _____
Date: 12-FEB-1987

Reviewer: S. M. Fisher, P. Palazzi
Comment: _____

Date: _____

ADAM9/DFP

The word 'is' appearing in a circle denotes a form of generalised relationship and here indicates that a 'Primitive' 'is' a 'CellArray' or 'Text' or a 'PolyLine' etc.

The diagram is described in detail by our data definition language (DDL). The complete DDL for the four diagrams of GKS is about 20 pages long and contains detailed comments, many of them taken from the original standard document. An extract from the DDL for the 'PICTURE' diagram describing two entity sets and a relationship between entities in those sets is shown above in Fig. 2.

Dataflow diagrams

A Dataflow diagram shows a system in terms of its active components (processes) as circles and the data interfaces between them (dataflows) as arrows. Each process may be decomposed into a further diagram to an arbitrary depth in order to reveal more detail. Dataflows may similarly be expanded in terms of their components <DEMARCO78>.

To couple DFDs and ER we require that data flows are ultimately expanded down to ER components i.e. entity-sets, attributes and relationships.

Figure 3 shows the top level diagram. Consider process 5 'View Picture', which represents the viewing pipeline. Its basic function is to take 'GraphicObjects' and 'SegmentTransformations' from the 'Segment Store', and 'GraphicObjects' directly from process 4 along with 'ViewingPipeline' information and 'BundleTables' from another store to produce 'DeviceDriverData'. Four further diagrams were drawn showing details of the system.

State Transition Diagrams

State transition diagrams are used to describe the states of a system, what event triggers a transition between states and what happens during that transition. For GKS only two such diagrams were needed. One is Fig. 13 of the GKS standard and deals with the five States of GKS itself and the other shows the setting of the Aspect Source Flags. To each diagram corresponds one state variable - itself part of the data model.

Algorithms

Transformations, such as the calculation of normalized device coordinates, are adequately described by a section of the FORTRAN code making use of variables corresponding to attributes of entities described in the data model, as shown in figure 4 below.

```
C  PROCESS  5.1 - Map to NDC space
C
C          Input  : REAL  WC_X,WC_Y   point in WC
C          Output : REAL  NDC_X,NDC_Y point in NDC
C
C          NDC_X = ( (WC_X-WrdWin_W(1,1) ) * WrdWin_Tn(1)) + WrdWin_N(1,1)
C          NDC_Y = ( (WC_Y-WrdWin_W(2,1) ) * WrdWin_Tn(2)) + WrdWin_N(2,1)
```

Figure 4: Normalisation transformation - specification by code

BUILDING A GKS SYSTEM FROM THE SPECIFICATIONS

An almost complete GKS-2D system was built from our specification, making use of ADAMO <QIAN87>. ADAMO is a software system to define ER data structures, map them onto tables and manipulate them from FORTRAN programs. The TAP (Table Package), the kernel of the ADAMO system, is a package of FORTRAN callable subroutines to manipulate ER and/or tabular data residing in memory. The TAP refers to special FORTRAN variables, that are automatically derived from the description contained in the DDL.

The manipulation of the data objects using the TAP takes place at the same level of abstraction as in the data definition, using about 15 subroutines corresponding to the elementary operations of the ER model. Object names are retained from DDL to code, and the programmer continues to think in terms of entities and relationships, following a simple path from the specification to the code.

A full level 2b system with FORTRAN binding was built, with the following omissions:

- Only those inquiry functions required by the test programs and benchmarks were written
- The only input routines coded are for locator input

A full metafile driver was written as well as a partial device driver for the Tektronix 4014. The specifications occupied a junior programmer for about 6 weeks and coding took a further four weeks.

Approximately half of the 201 external GKS subroutines deal with setting and inquiring individual attributes. They are completely trivial. For example, the routine GSPLI (Set Polyline index) has just two lines of normally executed code. The first line of executable code checks that the attribute 'State' of the 'Environment' is at least GKGOP (GKS open), and takes the form:

```
IF (.NOT.(Environment_State .GE. GKGOP)) GOTO error
```

The other line sets the current PolyLine bundle index to 'INDEX', the argument of the routine.

```
Environment_LnId = INDEX
```

The precise meaning of 'Environment_State', GKGOP and 'Environment_LnId' are defined in the DDL. The code of the corresponding inquiry function GQPLI (Inquire polyline index) is equally trivial.

The section of code shown in figure 5 is from the routine GPL (Polyline). GPL checks that the GKS state is at least Workstation active, and stores the polyline attributes if a segment is open. The device driver is then called.

In the extract below, after checking the GKS state the 'Primitive' is inserted into the entity set 'Primitive' by 'CALL INSENT(Primitive)'. INSREL relates the 'Primitive' to the open 'Segment'. The 'Primitive' is then inserted, and the call to INSGEN records that the 'Primitive' is a 'PolyLine'.

Figure 6 was produced by the printing routines of the TAP and shows an example of data held in the entity sets 'PolyLine' and 'LPoint'. The first five 'LPoint's belong to 'PolyLine' 1 and the other two to 'PolyLine 2'.


```

SUBROUTINE GPL (N,PX,PY)
.
.
C
C check current GKS state
C
      IF(.NOT.(Environment_State .GE. GWSAC))GOTO error
      IF(Environment_State .EQ. GSCOP)THEN
C
C create a Primitive and relate it to the current segment
C
          Primitive_ID = NEXT
          CALL INSENT (Primitive)
          CALL INSREL (Primitive,Primitive_Segment,Segment)
C
C create PolyLine and relate it to the new Primitive
C
          IF(Environment_Asf(1) .EQ. GINDIV)THEN
              PolyLine_LType = Environment_LType
          ELSE
              PolyLine_LType = INULL
          ENDIF
.
.
          PolyLine_ID = NEXT
          CALL INSENT (PolyLine)
          CALL INSGEN (Primitive,Primitive_Is,PolyLine)
      ENDIF
.
.

```

Figure 5: Extract from code - GPL

Table : PolyLine					ADAMO/TAP
ID	Index	LType	LWidth	LnCol	
1	=====	1	1.0	1	
2	=====	1	1.0	1	

Table : LPoint					ADAMO/TAP
ID	P(1)	P(2)	#PolyLine#	#WrdWin	
1	0.0	0.1	#	1#	2
2	0.0	0.24	#	1#	2
3	0.142857	0.24	#	1#	2
4	0.142857	0.1	#	1#	2
5	0.0	0.1	#	1#	2
6	0.0	0.1	#	2#	2
7	0.0	0.24	#	2#	2

Figure 6: PolyLine and LPoint tables

Timing tests were performed using a test program of our own which made use of the metafile and the WISS, a benchmark <DU86>, and our own data flow diagram plotting tool (no segments). As a result of these tests, minor changes were made to the ER schema and to the code to eliminate the obvious hot spots. This work took about 4 man weeks.

It then becomes difficult to quote figures and compare them with those of commercial packages, because each package executes rapidly for some operations, and slowly for others. Some packages also have tuning parameters which makes the study more complex. On average the execution speed of our GKS is comparable with the fastest commercial package we could find.

EXTENDING THE MODEL TO GKS-3D

The specifications were later extended to GKS-3D by adding four entity sets and changing the dimension of some attributes from 2 to 3. The dataflow diagrams for the viewing pipeline (process 5 and its expansion) were heavily modified by inserting the 'View transformation' and the 'Projection transformation'. See for example the paper presented at Eurographics '86 <SINGLET86>. The code was extended to test the viewing pipeline by drawing 3 dimensional wire frames.

CONCLUSIONS

With our approach:

- all data are described consistently in the DDL corresponding to the Entity-Relationship diagrams
- essential processing is shown in the data flow diagrams, where the data processed appears in the DDL
- distinct states are identified by state transition diagrams
- algorithms are defined by sections of code

This combination of Entity-Relationship, Data flow and state transition diagrams (and for transformations the code) is a powerful specification method which, partly because it makes much use of diagrams, makes these specifications easy to communicate and leads rapidly to a good design.

With an ER data manipulation language, the path from specification to a complete working system is straightforward and the result is not a throwaway prototype, but, as our timing results indicate a real program.

This approach has been tried for a range of applications and gives consistently good results. Its adoption by graphics systems designers would dramatically reduce development time and would allow standards to keep up with and thus be able to exploit new hardware.

ACKNOWLEDGEMENTS

We profited much from discussions with Ursula Berthon, Irene Seis, Jenny Tweed, Jurgen Bettels, David Myers, Andries Van Dam and Albert Werbrouck.

We are grateful to Friedrich Dydak and Mike Metcalf for their encouragement and support.

REFERENCES

- CHEN76 Chen, P. P., The Entity-Relationship Model - Toward a Unified View of Data. ACM Trans. Database Syst. 1 (1977), 9-36
- DEMARCO78 DeMarco, T., Structured Analysis and System Specification. YOURDON Press (1978)
- DU87 Du, S. GKS benchmark. Private communication
- DUCE86 Duce, D.A., Fielding, E.V.C., Better Understanding through Formal Specification. Comput. Graphics Forum 4 (1985) 333-348
- ISO82 ISO/DIS 7942 Information Processing - Graphical Kernel System (GKS) - Functional Description: GKS Version 7.2. ISO/TC97/SC5/WG2 N163 (1982)
- QIAN87 Qian, Z. et al., Use of the ADAMO data management system within ALEPH. Comput. Phys. Comms. 45 (1987) 283-298
- SINGLET86 Singleton, K., An Implementation of the GKS-3D/PHIGS Viewing Pipeline Eurographics '86