RAL-84-128

Rob

Science and Engineering Research Council
Rutherford Appleton Laboratory
CHILTON, DIDCOT, OXON, OX11 0QX

RAL-84-128

Better Understanding through Formal Specification

D A Duce and E V C Fielding

December 1984

Better Understanding through Formal Specification

D. A. Duce and E. V. C. Fielding

Rutherford Appleton Laboratory, Chilton, Didcot OXON OX11 OQX

ABSTRACT

The Graphical Kernel System (GKS) is now an ISO International Standard for computer graphics programming. One of the major innovations of the Standard is the bundled specification of aspects, a mechanism which gives the applications programmer the ability to tailor the appearance of a picture independently on each of the workstations on which it is displayed, using the capabilities of the workstations. GKS also incorporates the traditional method of individual specification of aspects in which each workstation does the best it can to represent global aspect values. In this paper a formal specification technique, the Vienna Development Method (VDM), is used to describe aspect specification. The GKS model of aspect specification is progressively constructed from simpler models. Properties of these simpler models are formulated and the specifications are proved to conform to these. The properties are then traced through the more complex models. The paper demonstrates the applicability of formal specification to the design of graphics software and the ability of formal techniques to catalyse the deeper understanding of designs.

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements Specifications; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs - *specification techniques*; I.3.4 [**Computer Graphics**]: Graphics Utilities; I.3.6 [**Computer Graphics**]: Methodology and Techniques - *languages*

General Terms: Design

Additional Key Words and Phrases: Abstract data type, abstract specification, attribute handling, bundled aspects, constructive specification, graphic data type, Graphical Kernel System, individual aspects, structural induction

Better Understanding through Formal Specification

D. A. Duce and E. V. C. Fielding

Rutherford Appleton Laboratory, Chilton, Didcot OXON OX11 OQX

1. Introduction

The aim of this paper is to show that the techniques of formal specification can be applied to the design of graphics software in a useful way. It will be shown that formal specification can give a succinct, readable and precise account of a design, and furthermore that there are techniques which enable properties of a design to be formulated and investigated. It will also, hopefully, be shown that this process can lead to a deeper understanding of the concepts involved, sometimes in unexpected ways.

The example used in this paper is attribute handling in the Graphical Kernel System (GKS) [1]. From simple beginnings, a specification is constructed which exhibits much of the behaviour of GKS and which sheds considerable light on the underlying concepts. Reasons of space prevent a more complete specification being developed here; the primary purpose of this paper is to illustrate techniques, not to give a complete specification of GKS.

The notation used in this paper is based on the Vienna Development Method (VDM) [2] of Jones. In order to minimize the introduction of notation, only a limited subset of VDM is used, and some simplifications have been made.

2. Overviews

2.1. GKS concepts

Pictures in GKS are constructed from a number of basic building blocks, called **primitives**. GKS has six output primitives: polyline, polymarker, text, fill area, cell array and generalized drawing primitive (GDP). For the purposes of this paper it suffices to consider a single primitive, polyline, which draws a connected sequence of line segments. Each primitive has an associated set of **parameters** used to define a particular instance of the primitive. In the case of polyline, the parameters are the coordinates of the vertices. Primitives can be grouped together in **segments**, but segments are not considered here.

GKS has introduced the concept of an abstract **workstation** to hide the peculiarities of device hardware. A workstation consists of zero or one display surfaces and zero or more input devices. An application program may direct output to more than one workstation simultaneously, but in this paper only one workstation is considered and input is disregarded.

Coordinate data in the parameters of a primitive are specified in **world coordinates** (WC), a Cartesian coordinate system. Transformation to the coordinate system of the display device is accomplished in two stages; firstly, world coordinates are transformed to an intermediate coordinate system called **normalized device coordinates** (NDC) by a window to viewport mapping termed a **normalization transformation**, then a second window to viewport mapping, called the **workstation transformation** transforms these coordinates to **device coordinates** (DC). The details of these transformations will not further concern us. It will be assumed in the specifications that polyline coordinate data are supplied in normalized device coordinates, and that the workstation transformation is fixed. Primitives can optionally be clipped to the boundary of the viewport of the normalization transformation, but clipping too, will be ignored.

The appearance of a primitive displayed on a workstation is determined by its parameters and additional data termed **aspects**. The aspects of a polyline are: linetype, which in GKS may be solid, dashed, dotted, dashed-dotted or implementation-dependent; linewidth scale factor, which is applied to the nominal linewidth provided by the workstation to give a value which is then mapped to the nearest available linewidth; and polyline colour index. For simplicity, it will be assumed in the specifications that follow that the value for linewidth can be specified directly, rather than as the product of a scale factor and a nominal width. It is also assumed that the workstation supports any linetype and linewidth requested, as, although it is a simple matter to map the requested value onto the nearest available value, this adds needless complexity for the present purposes. Colour will also not be considered.

The values of aspects are determined by **attributes**. There are two basic schemes for specifying aspects, termed **individual specification** and **bundled specification**. In the individual scheme the value of each aspect is determined by a different attribute; the linetype aspect by the linetype attribute and the linewidth scale factor aspect by the linewidth scale factor attribute. For each of the attributes there is an operation to set its value. In this scheme, the setting of the value of an attribute, such as linetype, applies to all subsequent polyline primitives until it is reset. For example:

Operation Sequence	**Linetype**	**Linewidth scale factor**
set_linetype (t_1)		
set_linewidth_scale_factor (w_1)		
polyline (pts)	t_1	w_1
set_linetype (t_2)		
polyline (pts)	t_2	w_1

Attribute values are bound to primitives upon creation and cannot subsequently be altered.

In the bundled mode of specifying polyline aspects, the values of all the aspects are determined by a single attribute, called the **polyline index**. A polyline index defines a position in a table, the **polyline bundle table**, each entry in which is termed a **bundle** and specifies the values for each of the aspects. The bundle corresponding to a particular polyline index is termed the **representation** of the index. There is an operation which sets the value of polyline index modally, as well as an operation to set the representation of a bundle index. When a polyline is created, the current value of the polyline index is bound to the primitive and cannot subsequently be changed. Bundles are bound to primitives when they are displayed. In GKS each workstation has its own polyline bundle table, which allows the application to control the appearance of polylines with the same polyline index independently on each workstation on which they are displayed, using the capabilities of the workstation. If a representation of a polyline index is changed, the appearance of polylines already created with that polyline index may also be changed to the new representation. Thus although the value of the polyline index with which a polyline is created cannot subsequently be changed, the representation with which the polyline is displayed can be changed. GKS admits that some workstations are able to perform such changes dynamically whilst others need to redraw the picture to effect the changes. GKS allows the application to control when such redrawing (regeneration) takes place. In this paper it is assumed that the workstation is capable of performing changes of polyline representation dynamically. In a previous paper [3] the regeneration mechanism of GKS is specified and its behaviour examined.

The simplified system described above forms a proper subset of a full GKS system.

Formal specifications for both of the schemes for specifying aspects are given in later sections and these specifications are developed into two alternative combined schemes.

2.2. The Formal Specification Technique

The purpose of a specification is to state **what** a system is to do, not **how** it is to do it. A formal specification defines a system in an implementation-independent way by the use of **abstract data types** to describe the internal state. An abstract data type is characterized only by the operations allowed over it.

The specification technique used in this paper, VDM, is an example of the **constructive** or **model-**

based approach, which models abstract data types in terms of mathematically tractable entities such as sets, lists and mappings. Other authors have considered the application of **algebraic** or **property-based** techniques to the specification of graphics software [4, 5, 6].

For our present purposes, a VDM specification has two components:

(1) a model of the state;

(2) operations over the state.

The first component describes the structure of the class of objects representing the state. If X is some class of objects, objects belonging to this class are said to have **type** X. Objects are built from basic objects (integers, reals etc.), tuples, sets, lists and mappings.

In the specifications given here, all operations have the general type:

$State \times Inputs \times State \rightarrow Boolean$

The effect of an operation in VDM is described by two predicates: a **pre-condition** and a **post-condition**. The former is a predicate over *State* and *Inputs* and defines the conditions under which the operation produces a valid result. The latter is a predicate over *State* (the initial state), *Inputs* and *State* (the final state), which defines the effect of the operation. Defining operations implicitly in this way, allows relations and thus non-determinacy to be specified. though non-determinacy does not arise in the specifications given here. Where the pre-condition is **true** (i.e. all values of the inputs and initial state produce a valid result), it is omitted.

The definition in the next section of some fundamental types, which are used in the subsequent specifications, provides the opportunity to illustrate and explain most of the notation used in this paper.

3. Basic Types

$NDC\_Points = \textbf{list of } NDC\_Point$
$NDC\_Point = \mathbf{R} \times \mathbf{R}$

$DC\_Points = \textbf{list of } DC\_Point$
$DC\_Point = \mathbf{R} \times \mathbf{R}$

$Bundle = Linetype \times Linewidth$
$Linetype = \mathbf{N}$
$Linewidth = \mathbf{R}$

$Polyline\_Index = \mathbf{N}$
$Polyline\_Bundle\_Table = \textbf{map } Polyline\_Index \textbf{ to } Bundle$

$elems : \textbf{list of } \alpha \rightarrow \textbf{set of } \alpha$
$elems(l) \triangleq \textbf{if } l = <\ > \textbf{ then } \{\ \} \textbf{ else } \{\textbf{hd } l\} \cup elems(\textbf{tl } l)$

$t : NDC\_Points \rightarrow DC\_Points$
$t(pts) \triangleq \cdots$

In the simplest case, a type has only one component, as in the definition of the type *NDC_Points*. It is defined to be a list, each of whose elements is an object of type *NDC_Point*. The type *NDC_Point* is the first example of a tuple, and is an ordered pair, each of whose components has the basic type **R** (real). Lists of points in DC coordinates are similarly defined by the types *DC_Points* and *DC_Point*.

The type *Bundle* is a (*Linetype*, *Linewidth*) pair, where a *Linetype* has the basic type **N** (natural numbers) and *Linewidth* is of type **R** (real).

An example of a mapping is the description of the polyline bundle table (used in the bundled aspects specification) as a mapping from a *Polyline_Index*, of basic type **N**, to a *Bundle*. A mapping is similar to a function except that it has a (possibly sparse) finite domain and the pairing of an element in the domain with an element in the range is constructed explicitly, rather than being defined by an expression.

Finally, the way in which a function is defined is shown, with the introduction of a function *elems*. The first line in the definition of *elems* is its signature, which characterizes the function by giving the types of its domain and range in terms of the generic type α. It states that *elems* takes an argument of type **list of** α and returns as result a **set** of objects of type α. The second line describes the effect of the function, which is to take a list and to produce a set containing the elements in the list. The operators **hd** and **tl** are used to obtain the head and tail of a list respectively, and $< \ >$ denotes the empty list.

Also introduced is a function t, which is used in the specifications to transform a list of points in NDC coordinates to a list of points in DC coordinates. As the exact definition of t is not of interest the details are not given here.

The notation for defining operations over the whole state will be introduced later. Definitions of types will not be restated in the following specifications, unless they have changed.

4. The Formal Specification of Individual Aspects

A formal specification of a simple model of GKS, with a single workstation and a single output primitive, polyline, must embody the concepts described in section 2.1. The concept of NDC space must be captured; some abstraction of the workstation concept must be made; and the aspect specification mode must be modelled. The combination of both the state definition and the definition of the operations over this state serves to capture these concepts.

Primitives are considered to be created in NDC space. (WC space has been disregarded for simplicity.) A suitable model for capturing the idea of a picture being built in NDC space is to have a component of the GKS state representing the NDC picture, to which is added each primitive as it is created.

The concept of a workstation can be simplified to the model of the picture in DC space that is displayed on the display surface. This DC picture is modelled similarly to the NDC picture, as a component of the GKS state to which each primitive is added as it is displayed. The operations over the state define the relationship between these two pictures, and show how they are constructed.

To model the individual mode of aspect specification, a state component for each attribute is required, representing its current value.

The definition of the state is now shown below, followed by some explanation.

The State

$$GKS = NDC\_Picture \times DC\_Picture \times Linetype \times Linewidth$$

$$NDC\_Picture = \textbf{list of } I\_NDC\_Polyline$$
$$I\_NDC\_Polyline = NDC\_Points \times Linetype \times Linewidth$$

$$DC\_Picture = \textbf{list of } I\_DC\_Polyline$$
$$I\_DC\_Polyline = DC\_Points \times Linetype \times Linewidth$$

States of the system are described by objects of the class *GKS*, which is defined to be a 4-tuple, with the first component of type *NDC_Picture*, the second component of type *DC_Picture* and with the last two components having the types of the attributes.

The NDC picture is modelled as a list of objects of type *I_NDC_Polyline* (the prefix '*I_*' standing for 'Individual'), which in turn is modelled as a list of points, a linetype and a linewidth. As polylines are created objects of type *I_NDC_Polyline* are formed, so the choice of the components of the type *I_NDC_Polyline* defines what is bound to the polyline at creation time.

The picture displayed on the workstation is also modelled as a list of objects, and at the DC level a polyline is described similarly by a list of points, a linetype and a linewidth. The choice of components for the type *I_DC_Polyline* defines what is bound to the primitive at display time. The DC picture description is a representation of the essential features of a polyline displayed on a workstation.

The use of the data type **list** in the definitions of the pictures in NDC and DC coordinate space allows the order in which primitives are created to be retained and used in the display of primitives. Had the data type **set** been used, this notion of order would have been lost. An order of display is not prescribed by the GKS document, but for the purposes of this illustration, the familiar model of preserving the order of creation in order of display is adhered to.

NDC Picture creation

One operation is allowed on an *NDC_Picture* component of the state - the creation of a new picture by adding a polyline to an existing picture. The definition of a function to do this is:

$$\boxed{\begin{array}{l} create\_i\_picture : NDC\_Points \times Linetype \times Linewidth \times NDC\_Picture \rightarrow NDC\_Picture \\ create\_i\_picture(pts, lt, lw, ndcp) \triangleq mk\_i\_ndc\_polyline(pts, lt, lw) :: ndcp \end{array}}$$

The effect of this function is to create a new object of type *I_NDC_Polyline* and to add it onto the list describing the existing NDC picture.

New objects are created by **constructor functions**. The name of a particular constructor function is taken from the type name of the object being created, prefixed with *mk_*. The operator *cons*, (denoted here by the symbol '::' which is used in infix form), adds an element at the head of a list.

DC Picture display

A similar function is defined over the *DC_Picture* component of the state:

$$\boxed{\begin{array}{l} display\_i\_picture : DC\_Points \times Linetype \times Linewidth \times DC\_Picture \rightarrow DC\_Picture \\ display\_i\_picture(pts, lt, lw, dcp) \triangleq mk\_i\_dc\_polyline(pts, lt, lw) :: dcp \end{array}}$$

Operations

Three operations over this state need to be defined:

polyline
set_linetype
set_linewidth

The operation definitions are preceded by a **let** clause which names the object *gks* and its components that comprise the initial state. The convention followed is that type names have capitalized initial letters and the names of instances are the lower case equivalents of the names of their types. The names of the final state and its components are the same as those of the initial state but decorated with a prime ('). Strictly these names should also be declared in the **let** clause, as follows:

$mk\_gks(ndc\_picture', dc\_picture', current\_linetype', current\_linewidth') = gks'$

but for conciseness this clause is omitted.

For simplicity, the **let** clause defining the *State* instances referred to in the pre- and post-conditions of the operations is defined once at the start of the specification of all the operations, rather than being repeated at the start of each operation definition. This should not cause any confusion.

The definitions of the operations are given below.

let $mk\_gks(ndc\_picture, dc\_picture, current\_linetype, current\_linewidth) = gks$ **in**

$polyline : GKS \times NDC\_Points \times GKS \rightarrow Boolean$
$polyline(gks, ndc\_points, gks') \triangleq$
post $ndc\_picture' = create\_i\_picture(ndc\_points, current\_linetype, current\_linewidth, ndc\_picture) \wedge$
$dc\_picture' = display\_i\_picture(t(ndc\_points), current\_linetype, current\_linewidth, dc\_picture)$

$set\_linetype : GKS \times Linetype \times GKS \rightarrow Boolean$
$set\_linetype(gks, linetype, gks') \triangleq$
post $current\_linetype' = linetype$

$set\_linewidth : GKS \times Linewidth \times GKS \rightarrow Boolean$
$set\_linewidth(gks, linewidth, gks') \triangleq$
post $current\_linewidth' = linewidth$

The first line of each definition gives the signature of the operation. The second line names the arguments and results of the operation, in the same sequence as in the signature; thus in the first two lines of the definition of *polyline*, *gks* is an object of type *GKS*, *ndc_points* is an object of type *NDC_Points* and the resulting object *gks'* is also of type *GKS*.

The effects of the *polyline* operation are defined in the post-condition which relates the initial and final states implicitly. The post-condition is written in terms of the previously defined functions *create_i_picture* and *display_i_picture* in order to facilitate comparison with later specifications of other attribute models which will redefine these subsidiary functions. It creates a new polyline in the NDC picture and displays this polyline in the DC picture. It can be seen from this definition that for each polyline that is added to the NDC picture, a corresponding polyline is also added to the DC picture.

Strictly one should also write:

$current\_linetype' = current\_linetype \wedge current\_linewidth' = current\_linewidth$

in the post-condition for *polyline*. By convention, the values of the components of the final state which are not given in post-conditions, are the same as the corresponding values in the initial state. The conventions used should not cause confusion, but it is necessary to appreciate their implications in order to understand the specification fully.

The effects of the post-conditions of *set_linetype* and *set_linewidth* are self-explanatory.

That concludes the specification of the individual aspect specification scheme, as well as the introduction of notation.

5. The Formal Specification of Bundled Aspects

In order to specify formally the bundled mode of aspect specification, once again, the concepts of the NDC picture and the DC picture must be modelled by components of the state in a way similar to that shown in the specification of the individual aspect scheme.

To model the bundled aspect scheme, there is only a single attribute (polyline index) whose current value has to be represented in the state. There is an additional concept to be captured; the idea of a polyline bundle table in which representations for polyline indices are stored. In this specification, as a polyline bundle table is associated with a workstation, the concept of a workstation is captured not only by the model of the DC picture, but also by the model of the polyline bundle table. The type used to describe a polyline bundle table was introduced in section 3, where basic types were defined and explained.

This specification exhibits the same fundamental structure as the specification for the individual aspects scheme.

The state of the system is described by objects of the class *GKS* defined as shown below.

The State

```
GKS = NDC_Picture × DC_Picture × Polyline_Bundle_Table × Polyline_Index

NDC_Picture = list of B_NDC_Polyline
B_NDC_Polyline = NDC_Points × Polyline_Index

DC_Picture = list of B_DC_Polyline
B_DC_Polyline = DC_Points × Polyline_Index × Bundle
```

As in the individual aspects specification, the NDC picture is modelled as a **list** of objects. However, the type of the objects in this list has been changed. A *B_NDC_Polyline* (the prefix '*B_*' standing for 'Bundled') is represented as a list of points and a polyline index, as only the polyline index is bound to the polyline at the time of its creation.

The DC picture, too, is modelled as a list of objects of a different type from the objects comprising the DC picture of the individual aspects scheme. The type *B_DC_Polyline* captures the concept of a bundle being bound to a primitive at display time. The need for polyline indices to be stored in the DC picture will become apparent later.

NDC Picture creation

The creation of a new polyline and its addition to the NDC picture is described by the function *create_b_picture*, which has a similar form to *create_i_picture*:

```
create_b_picture : NDC_Points × Polyline_Index × NDC_Picture → NDC_Picture
create_b_picture(pts, index, ndcp) ≜ mk_b_ndc_polyline(pts, index) :: ndcp
```

DC Picture display

The function *display_b_picture* which is defined over the DC picture for the bundled aspects scheme and corresponds to *display_i_picture* is:

```
display_b_picture : DC_Points × Polyline_Index × Bundle × DC_Picture → DC_Picture
display_b_picture(pts, index, b, dcp) ≜ mk_b_dc_polyline(pts, index, b) :: dcp
```

It describes the display of a polyline by its addition to the DC picture.

Operations

The operations in this system defined below are:

polyline
set_polyline_index
set_polyline_representation

```
let mk_gks(ndc_picture, dc_picture, polyline_bundle_table, current_polyline_index) = gks in

polyline : GKS × NDC_Points × GKS → Boolean
polyline(gks, ndc_points, gks') ≜
pre  current_polyline_index ∈ dom polyline_bundle_table
post ndc_picture' = create_b_picture(ndc_points, current_polyline_index, ndc_picture) ∧
       dc_picture' = display_b_picture(t(ndc_points), polyline_bundle_table(current_polyline_index),
                                        dc_picture)

set_polyline_index : GKS × Polyline_Index × GKS → Boolean
set_polyline_index(gks, polyline_index, gks') ≜
post current_polyline_index' = polyline_index
```

```
set_polyline_representation : GKS × Polyline_Index × Linetype × Linewidth × GKS → Boolean
set_polyline_representation(gks, polyline_index, linetype, linewidth, gks') ≜
post polyline_bundle_table' = polyline_bundle_table + [polyline_index → mk_bundle(linetype, linewidth)] ∧
     dc_picture' = recreate(dc_picture, polyline_bundle_table')

recreate : DC_Picture × Polyline_Bundle_Table → DC_Picture
recreate(dcp, pbt) ≜ if dcp = < > then < > else rebind(hd dcp, pbt) :: recreate(tl dcp, pbt)

rebind : DC_Polyline × Polyline_Bundle_Table → DC_Polyline
rebind(dcpl, pbt) ≜ let mk_b_dc_polyline(pts, index, b) = dcpl in
                        mk_b_dc_polyline(pts, index, pbt(index))
```

The form of the *polyline* operation mirrors that of the *polyline* definition in the previous specification, the differences in the binding of aspects being captured in the definitions of the types *B_NDC_Polyline* and *B_DC_Polyline* and of the functions *create_b_picture* and *display_b_picture*. Its effect is to create a new polyline in the NDC picture and to display it in the DC picture. For simplicity, the pre-condition ignores the fact that GKS defines a behaviour for the case when a representation of the polyline index is not defined.

The operation *set_polyline_index* resembles the operations *set_linetype* and *set_linewidth* of the individual aspects specification, and like them, sets the current value of an attribute (*polyline_index*).

The operation *set_polyline_representation* has two effects. The first line of its post-condition describes the addition to the polyline bundle table of the new representation specified for the polyline index. The operator '+' adds [*polyline_index* → *mk_bundle*(*linetype*, *linewidth*)] to the mapping, overriding any previous value associated with *polyline_index*. The second line of the post-condition describes the effect of *set_polyline_representation* on the DC picture, which is to change the representations of all polylines created with *polyline_index* to its new representation. Two auxiliary functions, *recreate* and *rebind* are used in the definition. *recreate* is defined recursively and traverses the DC picture applying the function *rebind* to each element in the list. *rebind* uses the *polyline_index* stored with each DC polyline to look up the associated representation in the new polyline bundle table, and then changes the representation of the polyline to this value. This explains why polyline indices are included in the representation of DC polylines. The notation *pbt*(*index*) in the definition of *rebind* denotes the application of the polyline bundle table mapping to a polyline index to yield the associated bundle.

6. A Comparison of the Individual and Bundled Aspect Specifications

We are now in a position to compare the individual and bundled systems just specified. This will illustrate some of the techniques that can be used to prove properties of specifications.

The first question to be posed is: what is the correspondence between the two systems - is there a sequence of operations in the one system that corresponds to a sequence of operations in the other? As both schemes of specifying aspects are concerned with the appearance of primitives at the DC picture level, and it is possible to define and use bundle representations with the same values as those used to set individual attributes, the relationship between the schemes should be expressible in terms of equivalence between the DC pictures that they produce. Firstly, then, this notion of equivalence needs to be defined. A polyline in the bundled DC picture will be said to be equivalent to a polyline in the individual DC picture if the result of the following function is **true**:

```
compare : B_DC_Polyline × I_DC_Polyline → Boolean
compare(b_dcpl, i_dcpl) ≜ let  mk_b_dc_polyline(b_pts, i, b) = b_dcpl
                          and  mk_bundle(b_lt, b_lw) = b
                          and  mk_i_dc_polyline(i_pts, i_lt, i_lw) = i_dcpl in
                               b_pts = i_pts ∧ b_lt = i_lt ∧ b_lw = i_lw
```

This definition says that the polylines have the same appearance if they have the same lists of vertices, linetypes and linewidths. A bundled DC picture is said to be equivalent to an individual DC picture (written '≡') if corresponding polylines in the pictures are equivalent in the above sense.

A sequence of operations in one system will be said to be equivalent to a sequence of operations in the other system if the resulting DC pictures are equivalent. This definition conveniently subsumes both static and dynamic behaviours.

The fact that the individual aspects scheme binds aspects to primitives on their creation, and that these cannot subsequently be altered, whilst the bundled scheme binds aspects to primitives upon display, and these can subsequently be altered, suggests that the bundled scheme is more powerful than the individual scheme. If the dynamic binding capabilities of the bundled scheme were not exploited (by not changing existing bundle representations), it could be used to achieve similar effects to the static binding permitted by the individual scheme. Stated in terms of equivalence at the DC picture level, it would seem reasonable that:

Property 1:

For every sequence of operations in the individual scheme, an equivalent sequence of operations can be given in the bundled scheme. //

How can this be shown formally?

A tool which will be required is the technique known as **structural induction** over lists:

To prove a property φ of lists, we prove:

(1) φ holds for the empty list, $< >$

(2) if φ holds for a list l, then φ holds for $e :: l$, and conclude that φ holds for **all** lists.

The basis for this method of proof is that all lists are built up from the empty list by prefixing. Strictly, proving (1) and (2) only guarantees that φ holds for **finite** lists, but infinite lists are not likely to arise in graphics hardware!

As an illustration of the application of this technique, a useful lemma that is required later will be stated and proved. This lemma states that if *recreate* applies to a DC picture a polyline bundle table which has the same representations of polyline indices as those present in the DC polylines comprising the picture, then the DC picture remains unchanged. More formally:

Lemma 1

$$\begin{aligned}&\mathit{recreate}(dcp, pbt) = dcp\\ &\text{iff } \forall\ \mathit{mk\_b\_dc\_polyline}(pts, i, b) \in \mathit{elems}(dcp)\ .\ pbt(i) = b\end{aligned}$$

Proof of Lemma 1:

By structural induction.

Base case: $dcp = < >$

It follows immediately that:

$$\mathit{recreate}(< >, pbt) = < >$$

Assume: $\mathit{recreate}(dcp, pbt) = dcp$

Add a polyline to each side and we have to show that:

$$\begin{aligned}&\mathit{recreate}(\mathit{mk\_b\_dc\_polyline}(pts, i, b) :: dcp, pbt) = \mathit{mk\_b\_dc\_polyline}(pts, i, b) :: dcp\\ &\text{iff } b = pbt(i)\end{aligned}$$

The lhs expands to:

$$\begin{aligned}
&recreate(mk\_b\_dc\_polyline(pts,\ i,\ b) :: dcp,\ pbt) \\
&\quad = rebind(mk\_b\_dc\_polyline(pts,\ i,\ b),\ pbt) :: recreate(dcp,\ pbt) \\
&\quad = mk\_b\_dc\_polyline(pts,\ i,\ pbt(i)) :: dcp && \text{by induction hypothesis} \\
&\quad = mk\_b\_dc\_polyline(pts,\ i,\ b) :: dcp && \text{if } b = pbt(i) \\
&\quad \neq mk\_b\_dc\_polyline(pts,\ i,\ b) :: dcp && \text{if } b \neq pbt(i)
\end{aligned}$$

which completes the proof. //

Property 1 can now be shown by proving the following statement:

Formal Statement of Property 1

Let b_gks^k denote a state of the bundled system, pbt^k denote the polyline bundle table component, and denote its DC picture component by $b\_dc\_picture^k$. Let i_gks^k denote a state of the individual system with DC picture component $i\_dc\_picture^k$. Suppose that $b\_dc\_picture^k \equiv i\_dc\_picture^k$, then if $i \notin \mathbf{dom}\ pbt^k$ or $pbt^{k+1}(i) = pbt^k(i)$, the following sequences of operations are equivalent (i.e. $b\_dc\_picture^n \equiv i\_dc\_picture^n$):

Bundled Operation Sequence	**Individual Operation Sequence**
$set\_polyline\_representation(b\_gks^k,\ i,\ t,\ w,\ b\_gks^{k+1})$	$set\_linetype(i\_gks^k,\ t,\ i\_gks^{k+1})$
$set\_polyline\_index(b\_gks^{k+1},\ i,\ b\_gks^{k+2})$	$set\_linewidth(i\_gks^{k+1},\ w,\ i\_gks^{k+2})$
$polyline(b\_gks^{k+2},\ pts_{k+2},\ b\_gks^{k+3})$	$polyline(i\_gks^{k+2},\ pts_{k+2},\ i\_gks^{k+3})$
...	...
$polyline(b\_gks^{n-1},\ pts_{n-1},\ b\_gks^n)$	$polyline(i\_gks^{n-1},\ pts_{n-1},\ i\_gks^n)$

These sequences of operations take each of the systems through a series of state transitions: $b\_gks^k \rightarrow b\_gks^{k+1} \rightarrow \cdots\ b\_gks^n$ in the bundled case, and similarly in the individual case through: $i\_gks^k \rightarrow i\_gks^{k+1} \rightarrow \cdots\ i\_gks^n$ (where $k+2 < n$). //

Since the polylines in each system are displayed with the same aspects (t and w), we would expect the two pictures to be the same.

Proof:

Inspection of the post-condition of *set_polyline_representation* reveals that this operation can modify the DC picture. A moment's reflection will suggest that the appearance of the DC picture will be unchanged by *set_polyline_representation* only if the representations of all the polyline indices used in the creation of the picture remain unchanged. This is the statement that was formalized and proved in lemma 1.

Case 1: $i \notin \mathbf{dom}\ pbt^k$

First it will be shown that the DC pictures are still equivalent in states b_gks^{k+2} and i_gks^{k+2}.

For the bundled system: in state gks^{k+1}, the polyline bundle table has the form:

$$pbt^{k+1} = pbt^k + [\ i \rightarrow mk\_bundle(t,\ w)\ ]$$

and the DC picture is changed in the following way:

$$b\_dc\_picture^{k+1} = recreate(b\_dc\_picture^k,\ pbt^{k+1})$$

Now: $\forall\ mk\_b\_dc\_polyline(pts,\ pi,\ b) \in elems(b\_dc\_picture^k)\ .$
$$pbt^{k+1}(pi) = pbt^k(pi) = b$$

since $pi \neq i$. Thus by lemma 1:

$$b\_dc\_picture^{k+1} = b\_dc\_picture^k$$

It follows immediately from the post condition of *set_polyline_index* that:

$$pbt^{k+2} = pbt^{k+1} \wedge b\_dc\_picture^{k+2} = b\_dc\_picture^{k+1}$$

And so: $b\_dc\_picture^{k+2} = b\_dc\_picture^{k}$

For the individual system: it follows immediately from the post-conditions of *set_linetype* and *set_linewidth* that:

$$i\_dc\_picture^{k+2} = i\_dc\_picture^{k}$$

Thus: $b\_dc\_picture^{k+2} \equiv i\_dc\_picture^{k+2}$

It is now shown that the *polyline* operation preserves the equivalence of the DC pictures by using structural induction over lists.

Base case: $b\_dc\_picture^{k+2} \equiv i\_dc\_picture^{k+2} = < >$

Perform a *polyline* operation in each system and then, from the definitions of the *polyline* operations, the DC pictures become:

$$b\_dc\_picture^{k+3} = mk\_b\_dc\_polyline(t(pts_{k+2}), i, mk\_bundle(t, w)) :: < >$$
$$i\_dc\_picture^{k+3} = mk\_i\_dc\_polyline(t(pts_{k+2}), t, w) :: < >$$

Now: $compare(mk\_b\_dc\_polyline(t(pts_{k+2}), i, mk\_bundle(t, w)), mk\_i\_dc\_polyline(t(pts_{k+2}), t, w)) = \mathbf{true}$

Thus: $b\_dc\_picture^{k+3} \equiv i\_dc\_picture^{k+3}$

Assume: $b\_dc\_picture^{n-1} \equiv i\_dc\_picture^{n-1}$

Perform another *polyline* operation in each system; and the DC pictures become:

$$b\_dc\_picture^{n} = mk\_b\_dc\_polyline(t(pts_{n-1}), i, mk\_bundle(t, w)) :: b\_dc\_picture^{n-1}$$
$$i\_dc\_picture^{n} = mk\_i\_dc\_polyline(t(pts_{n-1}), t, w) :: i\_dc\_picture^{n-1}$$

As before: $compare(mk\_b\_dc\_polyline(t(pts_{n-1}), i, mk\_bundle(t, w)), mk\_i\_dc\_polyline(t(pts_{n-1}), t, w)) = \mathbf{true}$

and so, by the induction hypothesis:

$$b\_dc\_picture^{n} \equiv i\_dc\_picture^{n}$$

which concludes case 1.

Case 2: $i \in \mathbf{dom}\ pbt^{k}$

In this case, if the new representation for i is the same as its old representation, then

$$pbt^{k+1} = pbt^{k} + [i \rightarrow mk\_bundle(t, w)] = pbt^{k}$$

Then by lemma 1,

$$b\_dc\_picture^{k+1} = b\_dc\_picture^{k}$$

and the proof then continues as for case 1.

However, if the new representation for i differs from the old representation, then:

$$pbt^{k+1} = pbt^{k} + [i \rightarrow mk\_bundle(t, w)] \neq pbt^{k}$$

and it follows, also from lemma 1, that:

$$b\_dc\_picture^{k+1} \neq b\_dc\_picture^{k}$$

and thus: $b\_dc\_picture^{n} \not\equiv i\_dc\_picture^{n}$

//

The next question to be asked is: what happens if bundled and individual specification modes are combined in a single system? The next sections consider two such systems.

7. The Formal Specification of Bundled or Individual Aspects

The next system to be considered is one in which all the aspects of each polyline may be specified either in an individual mode or in a bundled mode. The operations in this system are the combination of the operations in the individual aspects system and the bundled aspects system:

polyline
set_polyline_index
set_linetype
set_linewidth
set_polyline_representation

In addition, a new operation is added:

set_aspect_mode(*aspect_mode*)

The argument *aspect_mode* can take the values *BUNDLED* or *INDIVIDUAL*. If the argument has the value *BUNDLED*, subsequent polylines will be created in the bundled style, using the *current_polyline_index*. If the argument has the value *INDIVIDUAL*, the values of *current_linetype* and *current_linewidth* will be bound to the polyline when it is created. The operation *set_aspect_mode* may be invoked at any time.

The State

GKS = *NDC_Picture* × *DC_Picture* × *Polyline_Bundle_Table* × *Polyline_Index* ×
Linetype × *Linewidth* × *Aspect_Mode*

NDC_Picture = **list of** *NDC_Polyline*
NDC_Polyline = *I_NDC_Polyline* | *B_NDC_Polyline*

DC_Picture = **list of** *DC_Polyline*
DC_Polyline = *I_DC_Polyline* | *B_DC_Polyline*

Aspect_Mode = {*BUNDLED, INDIVIDUAL*}

In this *GKS* state, the NDC picture is again modelled as a list of objects of type *NDC_Polyline*. However, to accommodate the two modes of specifying aspects, the type *NDC_Polyline* is now defined as the disjoint union of the two types *I_NDC_Polyline* and *B_NDC_Polyline*. The implication of this is that an NDC picture may contain a mixture of objects of these types. The DC picture is defined similarly.

NDC and DC Picture functions

The operations on the NDC and DC pictures are the functions:

create_i_picture
create_b_picture
display_i_picture
display_b_picture

defined in sections 4 and 5.

Operations

The definitions of the operations in this combined individual or bundled mode system are:

```
let mk_gks(ndc_picture, dc_picture, polyline_bundle_table, current_polyline_index,
          current_linetype, current_linewidth, current_aspect_mode) = gks in

polyline : GKS × NDC_Points × GKS → Boolean
polyline(gks, ndc_points, gks') ≜
pre  ( current_aspect_mode = BUNDLED ⇒ current_polyline_index ∈ dom polyline_bundle_table )
post ( current_aspect_mode = INDIVIDUAL ⇒
            post polyline definition of the individual system in section 4. )
     ∧
     ( current_aspect_mode = BUNDLED ⇒
            post polyline definition of the bundled system in section 5. )

set_polyline_index : GKS × Polyline_Index × GKS → Boolean
set_linetype : GKS × Linetype × GKS → Boolean
set_linewidth : GKS × Linewidth × GKS → Boolean
  The definitions of the above three operations are as in sections 4 and 5.

set_aspect_mode : GKS × Aspect_Mode × GKS → Boolean
set_aspect_mode(gks, aspect_mode, gks') ≜
post current_aspect_mode' = aspect_mode

set_polyline_representation : GKS × Polyline_Index × Linetype × Linewidth × GKS → Boolean
set_polyline_representation(gks, polyline_index, linetype, linewidth, gks') ≜
post polyline_bundle_table' = polyline_bundle_table + [polyline_index → mk_bundle(linetype, linewidth)] ∧
     dc_picture' = recreate(dc_picture, polyline_bundle_table')

recreate : DC_Picture × Polyline_Bundle_Table → DC_Picture
recreate(dcp, pbt) ≜ if dcp = < > then < > else rebind(hd dcp, pbt) :: recreate(tl dcp, pbt)

rebind : DC_Polyline × Polyline_Bundle_Table → DC_Polyline
rebind(dcpl, pbt) ≜ case is_i_dc_polyline(dcpl): dcpl
                    case is_b_dc_polyline(dcpl): let mk_b_dc_polyline(pts, index, b) = dcpl in
                                                      mk_b_dc_polyline(pts, index, pbt(index))
```

The *rebind* function calls for some comment. The effect of the function on a DC polyline depends on whether the particular *dc_polyline* has type *I_DC_Polyline* or *B_DC_Polyline*. The function *is_X*(y) returns **true** if the object y is of type X and **false** otherwise. Thus the *rebind* function only affects polylines which were created in the bundled mode of working.

Behaviour

If the operation:

$$set\_aspect\_mode(gks^{k}, BUNDLED, gks^{k+1})$$

is invoked before the creation of any polylines, and is not reinvoked with the argument *INDIVIDUAL*, the behaviour of the system reduces to that of the bundled system defined in section 5. This can be seen by observing that the NDC and DC pictures produced will contain objects of type *B_NDC_Polyline* and *B_DC_Polyline* respectively, and that these types correspond to the types defined in the specification in section 5. Furthermore the operations *polyline*, *set_polyline_index* and *set_polyline_representation* reduce under this condition to the corresponding operations in section 5, and the operations *set_linetype* and *set_linewidth* have no effect on the NDC and DC pictures.

Similarly, if the operation:

$$set\_aspect\_mode(gks^{k}, INDIVIDUAL, gks^{k+1})$$

is invoked before the creation of any polylines, and is not reinvoked with the argument

BUNDLED, the system will behave as the system defined in section 4 for the individual mode of aspect specification. Note that the operation *set_polyline_representation* has no effect on a DC picture composed entirely of objects of type *I_DC_Polyline*.

These two cases do not exhaust the possible behaviours of this system, however, because the aspect specification mode can be changed dynamically, i.e. some polylines in the picture may be created in *BUNDLED* mode, and others in *INDIVIDUAL* mode.

One of the properties exhibited under these circumstances is: the representations of polylines created in the bundled mode can be changed as in the pure bundled system, but the representations of polylines created in individual mode cannot be changed, as in the case of the pure individual system.

The system behaves as a genuine combination of the bundled and individual systems. An extension of property 1 given in section 6 holds:

Property 2

For every sequence of operations in this system (including changes of aspect mode) an equivalent sequence can be given in the pure bundled scheme. Bundle indices already used in the bundled part of the picture cannot be reused in representing the individual part, otherwise the individual parts would undergo changes were their representations to be altered. //

8. The Formal Specification of Mixed Mode Aspects - GKS Style

Aspect handling in GKS is an extension of the system described in section 7. GKS allows a mixed mode of working in which some aspects of a primitive are determined from individual attributes. whilst others are determined by the bundled attributes.

Aspect source flags (ASF's) determine whether an individual or a bundled attribute value should be used for the corresponding aspect. In GKS there is one ASF for each aspect. Since the only aspects considered here are linetype and linewidth, only linetype ASF and linewidth ASF are needed. There are four possible combinations of aspect source flags:

Linetype ASF	**Linewidth ASF**
INDIVIDUAL	*INDIVIDUAL*
BUNDLED	*BUNDLED*
BUNDLED	*INDIVIDUAL*
INDIVIDUAL	*BUNDLED*

The NDC and DC polylines which are created and displayed under each of these cases are distinguished by their types.

The specification of the proper subset of GKS aspect specification handling is given below. The extensions follow a similar pattern to the extensions made in the development of the specification given in section 7.

The State

$\mathit{GKS} = \mathit{NDC\_Picture} \times \mathit{DC\_Picture} \times \mathit{Polyline\_Bundle\_Table} \times \mathit{Polyline\_Index} \times$
$\qquad \mathit{Linetype} \times \mathit{Linewidth} \times \mathit{ASF} \times \mathit{ASF}$

$\mathit{NDC\_Picture} = \textbf{list of } \mathit{NDC\_Polyline}$
$\mathit{NDC\_Polyline} = \mathit{I\_NDC\_Polyline} \mid \mathit{B\_NDC\_Polyline} \mid \mathit{Bt\_NDC\_Polyline} \mid \mathit{Bw\_NDC\_Polyline}$
$\mathit{Bt\_NDC\_Polyline} = \mathit{NDC\_Points} \times \mathit{Polyline\_Index} \times \mathit{Linewidth}$
$\mathit{Bw\_NDC\_Polyline} = \mathit{NDC\_Points} \times \mathit{Polyline\_Index} \times \mathit{Linetype}$

$\mathit{DC\_Picture} = \textbf{list of } \mathit{DC\_Polyline}$
$\mathit{DC\_Polyline} = \mathit{I\_DC\_Polyline} \mid \mathit{B\_DC\_Polyline} \mid \mathit{Bt\_DC\_Polyline} \mid \mathit{Bw\_DC\_Polyline}$
$\mathit{Bt\_DC\_Polyline} = \mathit{DC\_Points} \times \mathit{Polyline\_Index} \times \mathit{Linetype} \times \mathit{Linewidth}$
$\mathit{Bw\_DC\_Polyline} = \mathit{DC\_Points} \times \mathit{Polyline\_Index} \times \mathit{Linetype} \times \mathit{Linewidth}$

$\mathit{ASF} = \{\mathit{BUNDLED}, \mathit{INDIVIDUAL}\}$

NDC Picture creation

Uses *create_i_picture*, *create_b_picture* as defined in sections 4 and 5 respectively, as well as:

$\mathit{create\_bt\_picture} : \mathit{NDC\_Points} \times \mathit{Polyline\_Index} \times \mathit{Linewidth} \times \mathit{NDC\_Picture} \rightarrow \mathit{NDC\_Picture}$
$\mathit{create\_bt\_picture}(\mathit{pts}, \mathit{index}, \mathit{lw}, \mathit{ndcp}) \triangleq \mathit{mk\_bt\_ndc\_polyline}(\mathit{pts}, \mathit{index}, \mathit{lw}) :: \mathit{ndcp}$

$\mathit{create\_bw\_picture} : \mathit{NDC\_Points} \times \mathit{Polyline\_Index} \times \mathit{Linetype} \times \mathit{NDC\_Picture} \rightarrow \mathit{NDC\_Picture}$
$\mathit{create\_bw\_picture}(\mathit{pts}, \mathit{index}, \mathit{lt}, \mathit{ndcp}) \triangleq \mathit{mk\_bw\_ndc\_polyline}(\mathit{pts}, \mathit{index}, \mathit{lt}) :: \mathit{ndcp}$

DC Picture display

Uses *display_i_picture*, *display_b_picture* as defined in sections 4 and 5 respectively, as well as:

$\mathit{display\_bt\_picture} : \mathit{DC\_Points} \times \mathit{Polyline\_Index} \times \mathit{Linetype} \times \mathit{Linewidth} \times \mathit{DC\_Picture}$
$\qquad\qquad \rightarrow \mathit{DC\_Picture}$
$\mathit{display\_bt\_picture}(\mathit{pts}, \mathit{index}, \mathit{lt}, \mathit{lw}, \mathit{dcp}) \triangleq \mathit{mk\_bt\_dc\_polyline}(\mathit{pts}, \mathit{index}, \mathit{lt}, \mathit{lw}) :: \mathit{dcp}$

$\mathit{display\_bw\_picture} : \mathit{DC\_Points} \times \mathit{Polyline\_Index} \times \mathit{Linetype} \times \mathit{Linewidth} \times \mathit{DC\_Picture}$
$\qquad\qquad \rightarrow \mathit{DC\_Picture}$
$\mathit{display\_bw\_picture}(\mathit{pts}, \mathit{index}, \mathit{lt}, \mathit{lw}, \mathit{dcp}) \triangleq \mathit{mk\_bw\_dc\_polyline}(\mathit{pts}, \mathit{index}, \mathit{lt}, \mathit{lw}) :: \mathit{dcp}$

Operations

The operations in this system are again the combination of the operations in the individual aspect system and the bundled aspect system, with the addition of an operation to set the aspect source flags.

polyline
set_polyline_index
set_linetype
set_linewidth
set_polyline_representation
set_aspect_source_flags

```
let mk_gks(ndc_picture, dc_picture, polyline_bundle_table, current_polyline_index,
          current_linetype, current_linewidth, linetype_asf, linewidth_asf) = gks in

polyline : GKS × NDC_Points × GKS → Boolean
polyline(gks, ndc_points, gks') ≜
pre  ( linetype_asf = BUNDLED ∨ linewidth_asf = BUNDLED ⇒
          current_polyline_index ∈ dom polyline_bundle_table )
post ( linetype_asf = INDIVIDUAL ∧ linewidth_asf = INDIVIDUAL ⇒
          post polyline definition of the individual system in section 4. )
     ∧
     ( linetype_asf = BUNDLED ∧ linewidth_asf = BUNDLED ⇒
          post polyline definition of the bundled system in section 5. )
     ∧
     ( linetype_asf = BUNDLED ∧ linewidth_asf = INDIVIDUAL ⇒
          ndc_picture' = create_bt_picture(ndc_points, current_polyline_index, current_linewidth,
                                           ndc_picture) ∧
          let mk_bundle(lt, lw) = polyline_bundle_table(current_polyline_index) in
             dc_picture' = display_bt_picture(t(ndc_points), current_polyline_index, lt, current_linewidth,
                                              dc_picture) )
     ∧
     ( linetype_asf = INDIVIDUAL ∧ linewidth_asf = BUNDLED ⇒
          ndc_picture' = create_bw_picture(ndc_points, current_polyline_index, current_linetype,
                                           ndc_picture) ∧
          let mk_bundle(lt, lw) = polyline_bundle_table(current_polyline_index) in
             dc_picture' = display_bw_picture(t(ndc_points), current_polyline_index, current_linetype, lw,
                                              dc_picture) )

set_polyline_index : GKS × Polyline_Index × GKS → Boolean
set_linetype : GKS × Linetype × GKS → Boolean
set_linewidth : GKS × Linewidth × GKS → Boolean
  The definitions of the above three operations are as in sections 4 and 5.

set_aspect_source_flags : GKS × ASF × ASF × GKS → Boolean
set_aspect_source_flags(gks, lt_asf, lw_asf, gks') ≜
post linetype_asf' = lt_asf ∧ linewidth_asf' = lw_asf

set_polyline_representation : GKS × Polyline_Index × Linetype × Linewidth × GKS → Boolean
set_polyline_representation(gks, polyline_index, linetype, linewidth, gks') ≜
post polyline_bundle_table' = polyline_bundle_table + [polyline_index → mk_bundle(linetype, linewidth)] ∧
     dc_picture' = recreate(dc_picture, polyline_bundle_table')

recreate : DC_Picture × Polyline_Bundle_Table → DC_Picture
recreate(dcp, pbt) ≜ if dcp = < > then < > else rebind(hd dcp, pbt) :: recreate(tl dcp, pbt)

rebind : DC_Polyline × Polyline_Bundle_Table → DC_Polyline
rebind(dcpl, pbt) ≜ case is_i_dc_polyline(dcpl):   dcpl
                    case is_b_dc_polyline(dcpl):   let mk_b_dc_polyline(pts, index, b) = dcpl in
                                                       mk_b_dc_polyline(pts, index, pbt(index))
                    case is_bt_dc_polyline(dcpl):  let mk_bt_dc_polyline(pts, index, lt, lw) = dcpl
                                                   and mk_bundle(new_lt, new_lw) = pbt(index) in
                                                       mk_bt_dc_polyline(pts, index, new_lt, lw)
                    case is_bw_dc_polyline(dcpl):  let mk_bw_dc_polyline(pts, index, lt, lw) = dcpl
                                                   and mk_bundle(new_lt, new_lw) = pbt(index) in
                                                       mk_bw_dc_polyline(pts, index, lt, new_lw)
```

Properties of the Specification

Clearly, the specification exhibits all the properties of the system defined in section 7. However, the seemingly minor difference between the two systems introduces some totally new behaviour. The following property points to this:

Property 3

One might expect that all sequences of operations in this system would also have equivalents in the pure bundled system. However this is not the case. There are some sequences of operations for which equivalent sequences cannot be given in the pure bundled scheme. //

To see this, it is only necessary to consider the following example:

Operation Sequence	**Polyline Created**
$set\_polyline\_representation(i, t_b, w_b)$	
$set\_polyline\_index(i)$	
$set\_linetype(t_i)$	
$set\_linewidth(w_i)$	
$set\_aspect\_source\_flags(BUNDLED, INDIVIDUAL)$	
$polyline(pts)$	$mk\_bt\_dc\_polyline(t(pts), i, t_b, w_i)$
$set\_aspect\_source\_flags(INDIVIDUAL, BUNDLED)$	
$polyline(pts)$	$mk\_bw\_dc\_polyline(t(pts), i, t_i, w_b)$

where $t_i \neq t_b$ and $w_i \neq w_b$. There is no direct bundled equivalent of this picture, because the first polyline would require a bundle table entry:

$$i \rightarrow mk\_bundle(t_b, w_i)$$

whilst the second would require an entry:

$$i \rightarrow mk\_bundle(t_i, w_b)$$

where $t_b \neq t_i \wedge w_b \neq w_b$. This is clearly impossible. //

9. Conclusions

This paper has illustrated that formal specification can be used to express and analyse concepts. Our previous paper [3] in combination with this paper has laid the foundations for a complete specification of GKS.

The stepwise development of the GKS attribute handling model given here is the kind of process a designer might go through, and indeed there are parallels with the historical development of GKS within the ISO graphics working group. Development should proceed in a clean way in so far as the combination of concepts should not introduce untoward side effects. Part of the design process should be the isolation of properties possessed by concepts and the demonstration that these properties still hold in combination in a system. A further aid to understanding the behaviour of a system is to describe it in a specification notation which is executable. This approach is described in [7, 8].

In this paper we have chosen a property of aspect binding and traced it through the stepwise development of the GKS binding model and have shown which models exhibit the property. The fact that the last model does not, should prompt the designer to question why. A design decision has to be taken as to the importance of conformance to the chosen property.

The formal technique will not give an answer to these design questions directly, but it does help to clarify the issues involved and allows the designer to explore the design space more thoroughly.

References

1. *Graphical Kernel System (GKS) 7.2 Functional Description, ISO/DIS 7942,* Information Processing (4 November 1982).
2. C. B. Jones, *Software Development: A Rigorous Approach,* Prentice-Hall, Englewood Cliffs, NJ (1980).
3. D. A. Duce, E. V. C. Fielding, and L. S. Marshall, "Formal Specification and Graphics Software," RAL-84-068, Rutherford Appleton Laboratory, Chilton, Didcot, OXON OX11 0QX, U.K. (1984).
4. R. Gnatz, "An Algebraic Approach to the Standardization and the Certification of Graphics Software," *Computer Graphics Forum* **2**(2/3) (1983).
5. G. S. Carson, "The Specification of Computer Graphics Systems," *IEEE Computer Graphics and Applications*, pp. 27-41 (September 1983).
6. W. R. Mallgren, "Formal Specification of Graphic Data Types," *ACM Transactions on Programming Languages and Systems* 4(4), pp. 687-710 (October 1982).
7. P. Henderson, "Specifications and Programs," in *Software; Requirements, Specifications and Testing*, ed. T. Anderson, Blackwell Scientific Publications (To appear).
8. C. Minkowitz, "Specification to Prototype - A comparison of two formal methods of software design," Department of Computer Science, University of Stirling, Scotland (1984).