A TEXT, IMAGE, AND GRAPHICS EDITOR

AHMET COŞAR and BULENT ÖZGÜÇ
Bilkent University, Department of Computer Engineering and Information Science, Ankara, Turkey

1. INTRODUCTION

In this paper a system for the manipulation of text, image, and graphics items is presented. There are many products that manipulate text, image, or graphics data independently, but most of these systems are unable to manipulate them in a unified document [9, 11].

With the introduction of Smalltalk [10], Star [16], Lisa [19], Macintosh [18], and TEX [14] it has become possible to mix text and graphics data in a single document. The ODA/ODIF standard has also been developed for “the preparation, processing and interchange of text as used in office documents” [12]. The system that we have implemented is based on similar ideas but it allows the documents to be created using relational geometry [27], as well. Any change in a given data item causes the relational expressions existing in a document to be reevaluated and this makes our system an interactive WYSIWYG document editor [4, 8]. Currently, we are using this system for documenting various instances of animations generated by an animation package developed at Bilkent University [6]. We are also planning to enhance this system for storing animation sequence definitions, timings, etc. The relations are usually based on positional attributes, but facilities for defining relations on text valued attributes are also provided (Fig. 1).

When a user selects any one of the items on the screen a menu of the attributes of that item is presented to the user and the selection of the attribute to be updated is done through this menu. Insertion of new attributes to a selected item, especially for defining relations, can be done through a panel item as well. A detailed example for defining a “Bicycle” with relations is given in Section 3. Figure 2 shows a graphical object with its associated attributes and the data structures used for storing these attributes. For each data type (i.e., text, image, and graphics) there is a special display procedure (e.g., Graphics_Display($) Image_Display($) and Text_Display($)). That is called with the item to be displayed as its argument. This procedure uses the item’s attributes while displaying it on the screen. This type of an implementation is general and modular in nature and addition of new facilities to the system is greatly simplified. Definition of new item types, especially new graphics items, is possible and the method for this is discussed in Section 3.1.

A simple example of setting color of an item is discussed next to clarify the effects of the attributes. An attribute with the name Color and an integer value between 0 and 255 acting as an index to a colormap table are to be used. The C code that concatenates this information to the display parameter, pixel_operation is given below. The attribute structure is as defined in Fig. 2 and the pixel operation specifies a bitwise logical operation (like AND, OR, XOR, etc.) among the corresponding bits of the item to be displayed and the image in the frame buffer of the workstation as well as the color to be used for painting pixels. PIX_COLOR [23] is a system defined macro that is used for inserting color information into the pixel operation.

```
if (A= (struct attrib*)exist_attrib(item, "Color"))
    pixel_operation |= PIX_COLOR(A->attrval);
```

It is possible to define each attribute either by an integer value and/or a string expression. In order to allow users to define their relations using expressions, a simple language with special primitives is used. A simple relation is given below with an expression used for defining two items to have the same color.

```
Color='otheritemname'Color\ival
```

The quotation marks preceding words denote that those words are not evaluated and pushed into a stack and are later used by other commands as parameters. The backslash character precedes the commands of the language (as in the TEX convention [14]). The \ival command that stands for item value, pops two strings from stack. The first popped string is the attribute name and the second one is the item name that is unique for each item. The item and its corresponding attribute are searched by the system and the evaluated value of the attribute is pushed to the stack. Since evaluation of expressions with a cyclic relation set can cause infinite loops, checks are done by the system while evaluating expressions.

By setting various relations between different data items of any type, users will not have to deal with details that they have already defined. The implication of this is when a change in only one data item is made, all other data items might be updated accordingly if this is specified in the relation set. One such relationship is defined for inserting text into a text data that causes the rest of the line to be shifted right and if the right justification function is already defined for that text