Specifying Behavior and Semantic Meaning in an Unmodified Layered Drawing Package

James Fogarty[†], Jodi Forlizzi[†*], and Scott E. Hudson[†]
HCI Institute[†] and School of Design[*]
Carnegie Mellon University
Pittsburgh, PA 15213
{ jfogarty, forlizzi, scott.hudson }@ cs.cmu.edu

ABSTRACT
In order to create and use rich custom appearances, designers are often forced to introduce an unnatural gap into the design process. For example, a designer creating a skin for a music player must separately specify the appearance of the elements in the music player skin and the mapping between these visual elements and the functionality provided by the music player. This gap between appearance and semantic meaning creates a number of problems. We present a set of techniques that allows designers to use their preferred drawing tool to specify both appearance and semantic meaning. We demonstrate our techniques in an unmodified version of Adobe Photoshop®, but our techniques are general and adaptable to nearly any layered drawing package.

KEYWORDS: Visual specification, visual design tools, prototyping.

INTRODUCTION AND MOTIVATION
In order to create and use rich custom appearances, designers are often forced to introduce an unnatural gap into the design process. At one end of this gap, custom visual elements are often created by designers using a powerful layered drawing package, such as Adobe Photoshop or Adobe Illustrator®. These software packages provide designers with powerful tools, including layers and advanced filters, which allow designers to quickly and easily create rich custom appearances. At the other end of this gap, designers generally use a second tool to specify the relationship between these visual elements and a system. Common tools include Macromedia Director®, similar prototyping environments, and configuration files that are manually edited. For example, a designer creating a skin for a music player must separately specify the appearance of the visual elements in the skin and the mapping between these visual elements and the functionality provided by the music player. Several problems are created by this unnatural gap between the creation of an appearance and the specification of a corresponding semantic meaning. Most importantly, this gap means that many visual tasks, such as specifying the active region of a button, must take place in non-visual environments, such as with the Lingo® scripting language used in Macromedia Director. Further, this gap creates the requirement that designers learn the additional tools needed for specifying the desired semantic meaning.

Some tools, including Microsoft Visual Basic® and tools with similar functionality, support the specification of both visual appearance and semantic meaning. However, these tools place fundamental limits on designers. Because the creators of these tools often have neither the will nor the resources to duplicate the functionality of powerful layered drawing packages, designers find themselves working with drawing tools that are significantly less capable than those they prefer. These tools typically make it easy to create standard widget-based interfaces, but very difficult to create rich custom appearances. These limitations can undermine creative expression and may discourage designers from using these systems.

This paper presents a set of general techniques that allows designers to use their preferred layered drawing package, such as Adobe Photoshop, to associate behavior and semantic meaning with visual elements. *SLICE*, or Semantic Labeling in Convenient Environments, allows designers to work in an environment where they are comfortable and effective. These techniques support such tasks as the creation of a music player skin entirely within a layered drawing package. The techniques are extensible, do not require non-standard data in image files, do not require extra files be kept synchronized with image files, and support the association of arbitrary code with visual elements. We demonstrate our techniques in an unmodified version of Adobe Photoshop 6.0, but it is important to note that our techniques are general and adaptable to nearly any layered drawing program.

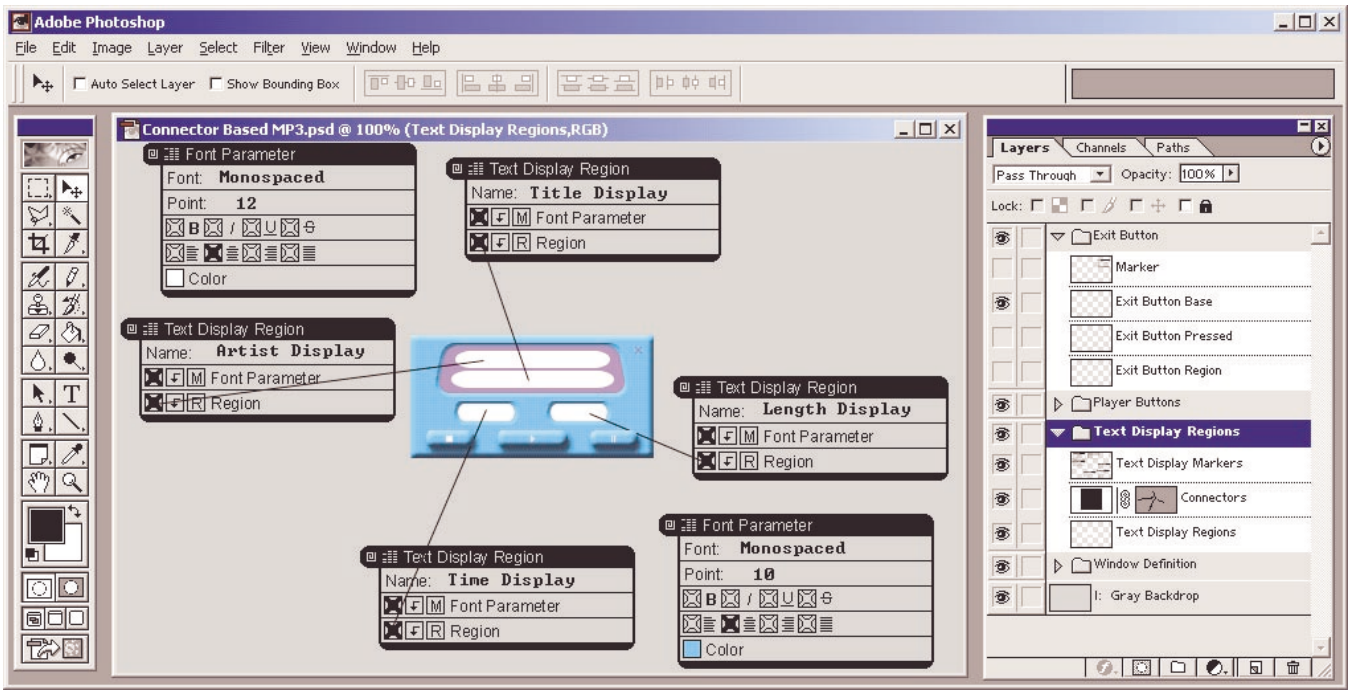

Figure 1. Using SLICE to create a music player skin.

Figure 1 presents an example of using SLICE to create a music player skin. This simple skin provides access to the play, pause, and stop functionality of the music player. It also specifies where to display the title of the current song, the artist of the current song, the current playback time, and the total length of the current song. Creating a music player skin entirely within Adobe Photoshop is an improvement on existing methods. Designers currently develop these skins by creating some number of image files and then manually editing a configuration file that links the functionality of the music player with these image files. This tedious task of cutting images into files and creating a matching configuration file is exactly the problem identified above, that designers must leave a visual working environment to specify the relationship between the image files and the music player.

This example demonstrates many of the basic ideas behind SLICE. Several *markers* are included in the layered image. These markers, which are visually similar to the marker shown in Figure 3, represent semantically meaningful objects. Using a natural interaction technique that is reminiscent of using a rubber stamp, markers are painted directly into the layered image document and exist as pixels in the layered image document. Each marker has several *fields* that have been provided. For example, the Text Display Region marker in the bottom-center has the value "Time Display" provided for a field named Name. Directly to the right of that marker, a Font Parameter marker has the value "Monospaced" provided for a field named Font, the value "10" provided for a field named Point, a checkbox field checked to indicate that the text should be centered, and a blue color painted into the box provided for a field named Color. Markers are also associated with *regions*, which are layers in the image document used to define areas to which markers apply. In this example, white regions have been used to mask the areas where text should be displayed in the skin. The black lines drawn between markers and regions are *connectors*, an advanced concept used here to specify relationships between markers and regions. This example also includes a demonstration of the use of spatial position to indicate relationships among markers. The document contains two groups, which each contains three markers. Within these spatial groupings, the Font Parameter markers indicate how text will be displayed in the Text Display Region markers. So this skin specifies that the current playback time and the length of the current song will be displayed with a blue 10 point font, while the artist and the title will be displayed with a white 12 point font. The light blue background, the purple display area, and the various buttons are all examples of *visual elements* that appear in the skin. For the sake of clarity, the markers that define buttons in this skin are currently not visible.

Figure 2. A music player using the skin in Figure 1.

Figure 2 shows a music player using the skin created in Figure 1. The player, constructed with jlGui [8], responds and provides playback information to the markers in the

document. For example, any Button marker named Play will start the music player. A Text Display Region named Time Display will be updated to display the current playback time. The music player accounts for a variety of markers and interacts appropriately with the markers that are present, ignoring the absence of any markers that were not used.

Music players also often provide an API for supporting visualizations of spectrum analyzer output. Our player supports any number of markers that inherit from a Spectrum Analyzer marker. The player provides each of these markers with spectrum analysis information as a song is played. Consider, for example, a visualization that flashes lights as the music plays. The marker for this visualization uses a region and two color fields. The creator of the skin specifies the region and paints the two colors into the marker. When playing, the marker paints the region a color that varies between the two provided colors according to the spectrum analysis information. This approach allows visualizations to be configured visually and allows new visualizations to be created as new markers.

In the next section, we will present an overview of SLICE documents. We will then present our document interpretation architecture and discuss our approaches to some of the problems encountered in interpreting SLICE documents. This is followed by a discussion of two more distinct uses for SLICE documents: the specification of aesthetic templates for the Kandinsky system [3] and a method of specifying animations. We then discuss some initial feedback gathered from sessions with designers. Finally, we compare SLICE to some related work and offer a short conclusion.

SLICE DOCUMENTS

A driving concern in our design of SLICE is the desire to remain entirely within a layered drawing package. This means that all necessary information needs to be stored in the pixels of an image and extracted from the image when needed. This also means that feedback and guidance are limited to what can be provided within the interface of the drawing package. Given these requirements, three major principles have guided the design of SLICE.

First, *individual layers in SLICE documents can be reliably interpreted with very simple image processing techniques*. By avoiding a need for complex gesture recognition and computer vision techniques, we minimize the opportunity for recognition errors. Simple image processing techniques are sufficient for several reasons. One important reason is that the alpha mask of each layer provides us with free and reliable segregation between the background and foreground of the layer. A second important reason is that we use several semi-structured interaction techniques. Semi-structured interactions techniques, which are much more flexible than tradition structured interactions, guide input in order to simplify recognition [1]. A third important reason is that we make extensive use of knowledge about the context in which we are interpreting a particular portion of the image. Using knowledge of context in recognition both allows for simpler recognition and allows identical arrangements of pixels to be interpreted differently according to their context [5].

Second, *SLICE documents are always organized from the top of a document to the bottom of the document*. In other words, any given layer in a document refers to, modifies, or labels only layers that are lower in the document. Creation is simplified because the person creating a document can use this general principle to decide where to place parts of the document. Interpretation is simplified because this organization removes many of the ambiguities that can exist when a given layer might refer to both layers above and below it.

Third, *SLICE documents dedicate a substantial number of pixels to ensuring that the document is human-readable*. This allows human use of documents to be based in recognition, rather than relying on recall. Because we cannot provide tooltips or other interface exploration devices inside the interface of a drawing package, the desired information is presented in the document. This is in contrast to an approach that minimizes the number of pixels used in specifying semantic relationships. We are comfortable with the use of extra pixels because the usability gain appears to be significant. Further, layered image documents allow entire sets of pixels to be made invisible when not needed.

Basic Types in SLICE Documents

We now discuss each of the basic types used in SLICE documents. As indicated in the introduction, documents are constructed from three basic types: *visual elements*, *regions*, and *markers*. Each marker contains several *fields*, which can be thought of as parameters to a function. These basic types are complemented by *connectors*, which are not required but are useful in some situations.

Visual elements are sets of pixels that define an appearance. Visual elements include the light blue background in our earlier example, the image that should be drawn when a button is in a pressed state, and a frame in an animation. Visual elements can also include appearances that are never actually drawn, such as an image that is used as input for a tiling or texture generation process.

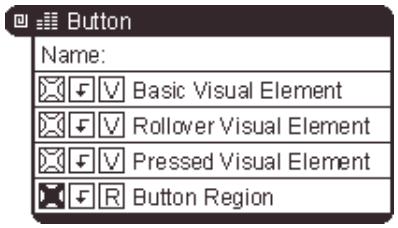

Figure 3. A marker indicating that several visual elements and a region represent a button.

Regions are defined by grayscale images that can be interpreted according to their use. In our earlier example, regions are used to indicate where the music player should display text. We can also use regions to specify the opaque areas of a non-rectangular clipping mask, to indicate where a click should be interpreted as a button press, and to indicate the path for an animated object to follow.

Markers define semantic objects and semantic relationships in a SLICE document. Our previous example includes markers that indicate regions should be interpreted as areas to display text. It also includes markers that indicate a font that should be used when displaying this text. Figure 3 shows a marker that indicates a set of visual elements and a region should be interpreted as a button. Note that each marker includes a human-readable type label and provides a human-readable list of the fields associated with the marker.

Fields provide a reusable framework for associating information with markers. Our button marker, for example, uses a text field to associate a name with the marker, three visual element fields to associate visual elements with the marker, and a region field to associate a region with the marker. Our previous example includes markers that use a color field, for which a color is specified by painting into the provided box. Another important field type is the marker field type, used when a marker references another marker. Taken together, reusable fields simplify the creation of new markers and provide a standard look across markers. They also localize the code for interpreting fields.

In the case of visual elements, regions, and marker field types, fields need to be associated with referents of the appropriate type. Because SLICE documents are organized from the top of the document to the bottom of the document, appropriate types need to be placed in the layers below the layer containing the marker. The fields in the marker are used to indicate the order that the referents should be placed below the marker. For our button marker shown in Figure 3, the first visual element below the button marker is interpreted as the default look of the button. The next visual element is interpreted as a rollover graphic and a third is interpreted as a presentation of the pressed state. A region specifies the clickable area of the button.

Fields can be either required or optional. For some field types, it is obvious whether or not a field has been provided. For example, if no text has been entered in a text field or no color has been painted in a color field, that field has not been provided. For the visual element, region, and marker field types, however, we use a checkbox to indicate whether the field has been provided. For required instances of these field types, the checkbox is already checked when the marker is placed. This allows the person creating the document to recognize that they must provide the field.

Connectors are an optional type that are not necessary for the creation of SLICE documents, but are useful in some situations. A connector is used when the person creating a document wants to override the default pairing between a field and its referent. This connector goes in a layer that is located between the marker and the referent. For example, Figure 1 includes a region layer that specifies all four of the areas where text is displayed. If a region field were paired with this entire region layer, it would be associated with all four of these areas. Because we want each display marker to be associated with only one of these areas, we have drawn connectors between the fields and the region. Each marker is associated with the part of the region that is contiguous with the endpoint of the connector. As a result, each display marker is paired with the mask for just one display area. Splitting each display area into its own region layer and positioning markers appropriately could achieve this same effect, but connectors allow a more concise specification and generally make SLICE more flexible. Multiple connectors can be attached to the same field if noncontiguous parts of the referent are required. Connectors in the same layer currently may not intersect, but this does not limit their descriptive power because any number of connector layers may be used.

Creating and Managing SLICE Documents

While SLICE documents are normal layered image documents, some techniques for interacting with these image documents make SLICE documents easier to create and manage. We will now discuss some of these techniques, focusing on how they work in Adobe Photoshop. It is worth noting that these techniques have been refined to work with the capabilities provided by Adobe Photoshop. An important step in the development of SLICE for use with different layered drawing packages will be the refinement of these techniques for those layered drawing packages.

Markers are painted into documents using custom brushes. Selecting a brush and clicking in the image paints the marker at the clicked location. Markers are defined as monochrome masks, and this technique should work for any drawing package that supports arbitrary custom brushes. If a drawing package cannot support arbitrary custom brushes, a simple copy and paste approach to marker placement would also be sufficient. Adobe Photoshop provides a simple mechanism for loading and unloading groups of brushes. This allows us to group related markers as a set of brushes that can be loaded and unloaded as needed. For example, a designer working on creating skins for a music player would load the appropriate set of custom brushes, while a designer working on specifying animations would load a different set of brushes. Adobe Photoshop also provides a simple mechanism for the creator of a brush to indicate how much the brush must be dragged before it creates a second print of the brush. This allows us to prevent designers from accidentally smearing a marker.

A related issue is how to edit the value of a field after a value has been provided. It is very straightforward to edit many types of fields. A color field, for example, can be edited by just painting a new color into the field. The behavior of the paint bucket tool gives the desired effect. For some field types, however, it could be difficult to restore the marker to the state it was in before the field was provided. Checkbox fields, in particular, are difficult to clear if they are filled with the color black. Doing so would required selecting and deleting the filled area of the checkbox without deleting the border that makes the checkbox behave appropriately when the paint bucket tool is applied. A useful strategy here is to fill the checkbox with a different color, such as gray or even purple. The filled area can then be easily selected for deletion without deleting the border of the checkbox. In the worst case, it is easy enough to just delete the entire marker, paint in a new marker, and provide the new desired field values.

Text fields also raise an interesting issue. Adobe Photoshop makes a distinction between text layers, which can still be edited with the text tool, and rendered text, which exists as pixels. Our document interpreter, therefore, allows either form of text to be used in a text field. Rendered text can be merged into the same layer as the marker, assuring that it will move with the marker and will always be positioned correctly. If left in a text layer, text can be positioned over a text field of a marker. In this form, the designer can more easily experiment with different values for the text field.

The ability to create groups of layers, referred to as layer sets in Adobe Photoshop, is helpful when managing large SLICE documents. As an example, consider the process of creating a button and exploring different positions for the button in an interface. The layers for the button, including the marker, the region, and any provided visual elements, can be placed inside a single layer set. All of these layers except one visual element can then be made invisible. The appearance of the button in the drawing package is then the same as the appearance of the button in the interface. Moving the layer set containing the button will maintain all of the appropriate relationships, and the designer can avoid the clutter that might arise from keeping all layers visible.

DOCUMENT INTERPRETER ARCHITECTURE

A document interpreter examines the pixels in a SLICE document and extracts the desired relationships among visual elements and markers. The document interpreter is currently implemented as an automation plugin for an unmodified off-the-shelf version of Adobe Photoshop. Using an automation plugin allows us to include a command to interpret a SLICE document in the main menu bar. It also allows us to automatically save the document before we begin, automatically flatten complex layers, and revert the document to its original state when we finish interpreting it. Our document interpreter uses the Adobe Photoshop automation API to access the pixels in each layer of the document and then manipulates the layers as collections of pixels. As such, all of our recognition and interpretation code is completely independent of Adobe Photoshop. If adapting SLICE to a layered drawing package that does not support the functionality needed for our current automation approach, it would be sufficient to use a second program that either interpreted exported layer images or extracted layer information from the file format used by the drawing package. We now present each step in the document interpretation process.

Marker Recognition

Markers are recognized in a two-step process. They are first located and then identified. They are located by searching in an image for a pixel pattern that is unlikely to appear for any other reason (the next section indicates how to handle the case where this pixel pattern is used for some other reason). At fixed offsets from the location of this pattern, a set of regions is examined to determine which are active. Our current patterns are presented in Figure 4. These regions are part of the marker brush and are filled in when the marker is created, not each time the marker is used. Knowing which regions are active provides us with a unique identifier of the marker type. We currently use 15 regions, giving us 32,768 possible types of markers. This is much more than we currently need and very simple extensions would allow the number of markers to be made arbitrarily large without requiring any change to existing markers.

Once a marker has been located and identified, the identity of the marker is used to access the marker specification. These marker specifications are installed together with the corresponding marker brushes. This marker specification lists the fields in the marker, the type of each field, the offset from the marker location to the pixels to be interpreted for that field, and any additional information needed to interpret the field.

With this field offset information available, the parsing of individual fields is delegated to a class for the field type. Checkbox fields determine if they have been checked, color fields extract the provided color, and text fields use an OCR engine [18] to recognize text rendered into the marker. Fields with references do not yet find their referents. These references are resolved in a second pass through the document discussed below.

Figure 4. Our current marker locator pattern and identifier pattern. Bits three and fifteen are active.

Layer Type Recognition

Layer types can be explicitly set with a prefix at the beginning of the layer name. Any characters before a colon in the name of a layer are considered a prefix. For example, a layer that is named "Visual Element: GrnBtn", "Visual: GrnBtn", or "V: GrnBtn" will be interpreted as a visual element regardless of the contents of the layer. This allows layers that look like they contain markers to be treated otherwise. Prefixes are recognized for each layer type, and a special "Ignore" or "I" prefix can be used to indicate that the document interpreter should completely ignore a layer. We also support the use of an ignore prefix on a layer set, indicating the every layer in the set should be ignored.

If a layer prefix is not provided, the contents of a layer are examined in order to heuristically determine the layer type. Our current heuristics are very simple. If a layer contains any markers, it is labeled as a marker layer. Non-marker layers that contain color are labeled as visual elements. If an image contains neither markers nor color, the document interpreter applies our connector recognition techniques to determine if the active pixels in the layer are consistent with a connector layer. If the active pixels are not consistent with a connector layer, the layer is labeled as a region. These heuristics seem to work well in practice. We have only found it necessary to explicitly label grayscale visual elements.

Connector Recognition

A connector layer consists of non-intersecting straight lines. Lines are currently extracted by finding the two most distant points in each connected set of active pixels. If a set of connected active pixels forms a line, the resulting pair of points will be the endpoints of the line. We check that a line drawn between the two points would account for all of the pixels in the connected set. If the layer has been explicitly labeled as a connector layer, we accept the endpoints regardless of whether the line accounts for every pixel in the set. If no explicit labeling has been provided, the failure of the line to account for the active pixels in the set indicates that the layer is not a connector layer.

An important parameter to this process is the width of the line that is used when determining if a line accounts for the pixels in the set. If the line is too narrow, some connected sets that form lines will be rejected because they are wider than the line being used. If the line is too wide, small oval and rectangular regions will be misidentified as connector lines. We currently use a line width of five pixels, which is relatively narrow. We have opted for a narrow line because connectors are an advanced feature in SLICE documents. Requiring that people experienced with SLICE sometimes explicitly label their connector layers is preferable to people just beginning with SLICE documents being unable to determine why their small regions are being misidentified.

It is also worth noting that our line extraction technique is much less complex than typical advanced computer vision techniques. It is, however, preferable for the situation in which we are using it. Because the alpha mask of the layer provides us with perfect segregation between the pixels in the line and pixels not in the line, our problem is only to fit a line to these pixels. For example, a standard Hough Transform [6] is inappropriate because we require the endpoints of the line. We experimented with a Progressive Probabilistic Hough Transform [14], but found that it often incorrectly recognized long narrow lines as several shorter lines. Increasing the minimum line length parameter to the algorithm resulted in the algorithm missing short lines. We are interested in exploring other techniques that may allow us to permit intersecting connector lines, but are generally satisfied with our current solution.

Pairing Markers with References

After the pixel contents of each layer have been recognized, a second pass is made through the document to pair markers with their referents. This straightforward process is based on the use of connectors and the proximity of the appropriate types to the marker.

In the absence of connectors, visual elements and regions are paired with fields according to the result of a search through the layers below a marker. The first visual element layer encountered is paired with the first visual element field in the marker. Additional visual elements are found by searching the layers below that visual element. The same process is followed for regions for which a connector has not been provided. This pairing process does not require that visual elements or regions be paired with only a single marker. In fact, it is common for multiple markers to reference the same visual element or region. This pairing process also explicitly allows some looseness in the placement of regions and visuals elements used by a marker. For example, our button marker expects up to three visual elements followed by a region. Our pairing process will work correctly if this region is instead placed before the visual elements or even between two of the visual elements.

Markers that refer to other markers without using connectors are paired according to a very similar approach. The first difference is that markers can appear in the same layer as the marker to which they refer. The second difference is that more than one marker can appear in a given layer. Given these differences, the search is performed in much the same way as for visual elements and regions. If there is more than one marker of the desired type in the layer found by the search, the document interpreter selects the marker whose (X, Y) location is closest to the field with which the marker is being paired. As an example, Figure 1 contains Text Display Region markers that are paired with Font Parameter markers in the same layer according to their spatial proximity.

Connectors override the pairing approach just described. The document interpreter instead searches the layers above the connector for the first field that one end of the connector points to. It uses the type of the field at that end of the connector, called the origin end of the connector, to search the layers below the other end of the connector, called the target end of the connector, for the first valid target. In the case of a region or visual element, this downward search ends at the first appropriately typed layer with a nonzero alpha value at the target end of the connector. The search yields the set of connected pixels with nonzero alpha values. If multiple connectors are associated with the same region or visual element field, the pixels that each of the connectors point to are combined. In the case of a connector used to indicate a pairing with a marker, the downward search ends at the first appropriately typed marker located at the target end of the connector.

An advanced aspect of marker pairing is our support for marker inheritance. Marker type specifications can include a list of marker types from which the type inherits. The marker can then replace any of those marker types as a field in another marker. Marker inheritance is useful in a variety of situations. For example, windowing systems usually include a Frame type that inherits from a Window type. We have created a Window Drag Region marker type that allows a user to move a window by holding the mouse down within the region and dragging the window. Because our Frame marker type inherits from our Window marker type, a Window Drag Region marker also works with a Frame marker. Marker inheritance also proves useful when a marker requires many fields to completely specify. A Font marker type, for example, requires a number of parameters to completely specify. It is often sufficient, however, to provide only a font name, point size, and color. A Simple Font marker type inherited from the basic Font marker type allows default values to be automatically provided for the values that are only occasionally changed.

Exporting SLICE Documents

Once the document interpreter has interpreted a SLICE document, it is exported as an XML document and a set of accompanying image files. This XML document includes the markers found in the document, the fields associated with each marker, and the necessary information for accessing the exported images. This interpreted form is intended for deploying a SLICE document in a runtime environment, and it is completely independent of the layered image document from which it is generated. As such, this interpreted form can be moved or deleted with no impact on the SLICE document used to generate it.

We have created a small toolkit, implemented in Java, for loading, displaying, and interacting with interpreted SLICE documents. This toolkit centers on a document class that reads an XML file, instantiates the marker types specified in the XML file, and provides each marker with its fields. The document class provides mechanisms for markers to listen to interface events and for applications to listen for events generated by markers. As illustrated with our music player example, this toolkit provides a simple mechanism for connecting markers. Applications can interact with semantically meaningful objects, such as responding to button click events generated by a Button marker or setting the text displayed in a Text Display marker.

EXAMPLES

This section presents two uses of SLICE documents that are distinct from our earlier music player skin example. We first present the use of SLICE documents to create aesthetic templates for use with the Kandinsky system. We then demonstrate techniques for creating an animated image that displays a series of frames as it moves along a path.

Kandinsky Aesthetic Templates

Figure 5 shows a simple aesthetic information collage generated by the Kandinsky system [3]. Aesthetic information collages provide an aesthetic contribution to the space in which they are displayed, but also convey information. As such, they share many of the motivations presented in [17]. Kandinsky generates aesthetic information collages by combining an aesthetic template, which is an image-based expression of an artistic intent, with images selected to represent information. Aesthetic templates specify both visual elements of a collage and criteria used to guide placement of images in the collage. In this simple example, the aesthetic template is based on an image of five paint bottles, each a different color. The aesthetic template identifies each bottle as a region into which collage images can be placed. It also provides criteria that guide the placement of images into these regions. In this example, images are placed into regions so that they match the color of the paint bottle. They are also arranged in a 'V' shape that complements the arrangement of the paint bottles. A detailed discussion of this template and the process for generating aesthetic information collages is provided in [3].

Figure 5. A simple aesthetic information collage generated by the Kandinsky system.

Figure 6 presents two views of the SLICE document used to create this aesthetic template. The portion of the SLICE document that is shown in the top view specifies the collage generation criteria for three of the five bottle areas in this aesthetic template. The most important markers used are the Image Region markers, which are in the layer named Markers. The Image Region marker in the bottom-right corner specifies that Kandinsky should place a collage image in the area corresponding to the green bottle. Directly above this marker is an Image Region Color Criteria marker. A green color has been painted into the color field of the marker, indicating that the collage image region will attract green images during the collage generation process. The spatial positioning of markers is used to indicate the desired pairing between Image Region Color Criteria markers and the appropriate Image Region markers. Placed in the layer named Balance Point Markers are Image Region Balance Criteria markers used to specify the 'V' shape desired when images are placed into the collage image regions. The Balance Point Region fields of these markers are paired with the small circular regions located in the layer below the markers. As with the Image Region Color Criteria markers, spatial positioning is used to indicate the desired pairing between Image Region Marker fields and the appropriate Image Region markers. The layers named Gray Backdrop are provided only to allow visibility of the balance point region in this printed document, and the "I" prefix on the layer names indicates that the document interpreter should ignore these layers.

Note that the collage image regions for the other two bottles are also defined in this same SLICE document, in the layer sets named Purple Bottle and Blue Bottle. The second view of this document shows the layers in Purple Bottle. They are arranged in the same order as the layers in the top view, with the exception that only one bottle region is defined in the region layer and connectors are not used. Note that all five of the paint bottle image regions in this template could be specified in this way, allowing a beginning user of SLICE documents to construct this document without using connectors. Further, the purple and blue paint bottle regions intersect. Because these regions intersect, connectors could not be used to distinguish between them if they were in the same layer. For this case, these regions need to be specified in different layers.

Specifying Animations

Our final example demonstrates a technique for creating simple animations. An Animation marker is placed in the document. The marker contains a Name field, a Start Time field, and a Duration field. A designer can specify that an animated object should cycle through a series of images by adding any number of Animation Frame markers. A designer can similarly specify that the animated object move along a path by drawing the path and using an Animation Path marker to associate the path with the Animation marker.

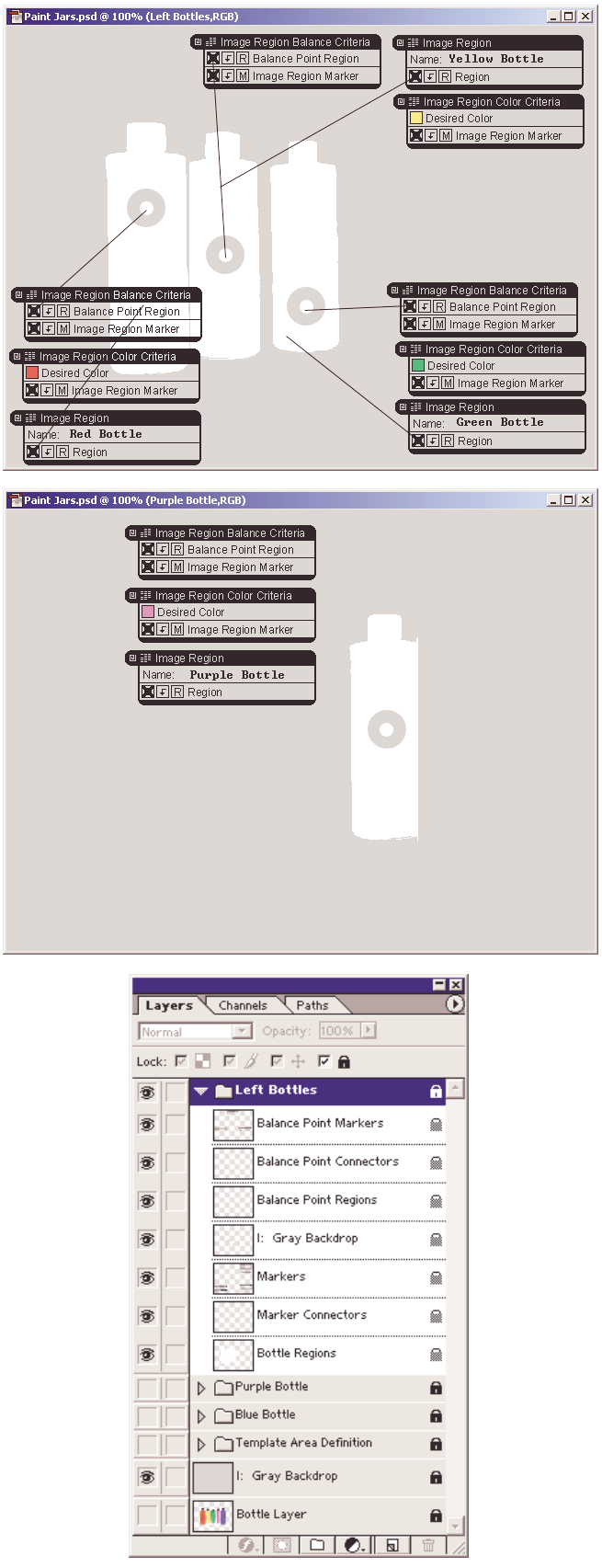

Figure 6. Two views of using SLICE to create a simple aesthetic template for the Kandinsky system.

Figure 7 illustrates this approach to creating simple animations. This is a simple animation that moves a pulsating yellow ball along a curved path. The pulsating effect is achieved by toggling between two frames every 500 milliseconds. The path animation is achieved by creating a region layer containing the desired path. Another region is used to indicate which end of the path is the starting end. The Animation object is moved along this path by translating the graphics context in which the Animation is drawn. Although a default linear timing function is used in this example, different timing functions are available. For example, a slow-in, slow-out timing function is often useful [10]. Custom timing functions can also be defined by using a marker that takes the location of a (0, 0) coordinate, the location of a (1, 1) coordinate, and a line defining a function from (0, 0) to (1, 1) [7].

The details of this example are less important than the marker creation strategy that it illustrates. We have already presented inheritance as a mechanism for managing large or complex markers. This example illustrates another approach for managing complex markers, one that is based on the Decorator design pattern [4]. The Animation marker itself has very little functionality, but provides a central object to which many other markers can be attached. This strategy for creating related sets of markers allows sets of simple markers to be created instead of large and complex markers. As new functionality is desired, new markers can be added to the set. We could, for example, loop our animation by adding an Animation Loop marker.

DESIGNER FEEDBACK

In order to inform the continuing development of SLICE, we informally presented SLICE to some designers that use Adobe Photoshop on a regular basis. We discussed the ideas behind SLICE, explained how SLICE documents work, and showed them the document for the music player skin shown in our introduction. We then had each designer create a skin that was at least as complex as our example skin, allowing them to use our example skin as a reference.

By the end of this ninety-minute session, the designers understood and were comfortable with SLICE documents. They could place markers, provide values for fields, and create relationships between markers, visual elements, and regions without guidance and without using our example skin as a reference. They found our guiding organizational principle, that any given layer in a document refers to, modifies, or labels only layers that are lower in the document, to be helpful in understanding the layout of a SLICE document. While none of the designers had previous experience creating a skin for a music player, they all drew connections between this task and other tasks requiring the combined used of Adobe Photoshop and another tool. They appreciated the ability to complete the task in a single tool, and we observed them integrating SLICE into their existing techniques for managing Adobe Photoshop image documents. For example, designers often

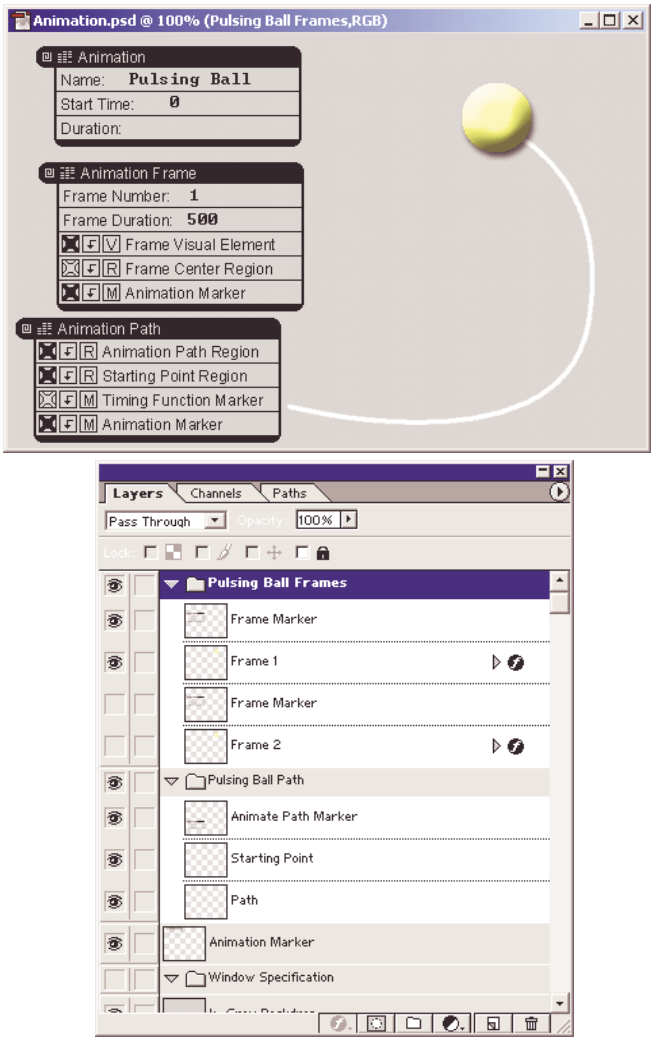

Figure 7. Specifying a simple animation sequence.

create multiple instances of similar objects by creating one instance, duplicating it as needed, and then making the desired changes. We observed them using this same technique to create similar SLICE objects, as in the case of multiple buttons with a similar appearance. The designers felt that the successful use of SLICE documents requires strategies for managing layers, such as grouping related layers into layer sets and giving meaningful names to layers. They also felt that this was true of any large Adobe Photoshop document and not limited to SLICE documents. We are pleased with these initial responses to SLICE.

RELATED WORK

Our work shares some of the same motivations as sketching based specification systems such as SILK [9] and DENIM [12]. Though we take very different approaches to the problem, both SLICE and these systems are interested in making it easier for designers to create and explore prototypes in a more natural environment. Specifically, SLICE is designed for use at a later point in the design process than SILK and DENIM. SLICE supports the creation of high-quality, finished appearances, and is not

intended to replace sketching. Similarly, we share some of the same motivations as DEMAIS, which uses multimedia storyboards to enable designers to explore multimedia applications [2].

A recent enhancement to SILK and DENIM allows the definition of reusable components in a sketching environment [13]. These components are limited to the storyboarding approach taken by these systems. Unlike our markers, they do not support arbitrary functionality. While well suited to the sketching context in which they are used, they differ substantially from our approach.

Pierce and Pausch present a technique for identifying active parts of a 3D model by using painted images similar to the images used to assign texture to a 3D model [16]. This relates to our ideas on specifying appearance and semantic meaning in the same environment. However, they present a very limited case and require a separate indication of what behavior should be associated with a painted region.

CONCLUSION

We have presented SLICE, a set of general techniques that allow designers to specify both visual appearance and semantic meaning in their preferred layered drawing package. The techniques used by SLICE are extensible, do not require non-standard data in image files, do not require extra files be kept synchronized with image files, and support the association of arbitrary code with visual elements. Currently implemented in an unmodified version of Adobe Photoshop 6.0, our techniques are general and adaptable to nearly any layered drawing package.

ACKNOWLEDGMENTS

We would like to thank designers Matthew Mowczko, Emma van Niekerk, and Arie Stavchansky for their feedback. We would also like to acknowledge libpng, zlib, jlGui, and the Open Source Computer Vision Library, all of which have been used in our document interpreter or our examples. This work was funded in part by the National Science Foundation under Grant IIS-0121560 and the first author's NSF Graduate Research Fellowship. All trademarks are the property of their respective owners.

REFERENCES

1. Avrahami, D., Hudson, S., Moran, T. P., and Williams, B., "Guided Gesture Support in the Paper PDA", *Proceedings of the 2001 ACM Symposium on User Interface Software and Technology (UIST 2001).*
2. Bailey, B. P., Konstan, J. A., and Carlis J. V., "DEMAIS: Designing Multimedia Applications with Interactive Storyboards", *Proceedings of the 2001 ACM Conference on Multimedia (Multimedia 2001).*
3. Fogarty, J., Forlizzi, J., and Hudson, S., "Aesthetic Information Collages: Generating Decorative Displays that Contain Information", *Proceedings of the 2001 ACM Symposium on User Interface Software and Technology (UIST 2001).*
4. Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995.
5. Gross, M., and Do, E.Y.-L., "Demonstrating the Electronic Cocktail Napkin: A Paper-Like Interface for Early Design", *CHI Letters: Proceedings of the 1996 SIGCHI Conference on Human Factors in Computing Systems (CHI 1996).*
6. Hough, P. V. C., "Methods and Means for Recognising Complex Patterns", U.S. Patent 3 069 654, Dec 1962.
7. Hudson, S., and Stasko, J. T., "Animation Support in a User Interface Toolkit: Flexible, Robust, and Reusable Abstractions", *Proceedings of the 1993 ACM Symposium on User Interface Software and Technlogy (UIST 1993).*
8. *jlGui - Java Music Player*, Web Page: http://www.javazoom.net/jlgui/jlgui.html.
9. Landay, J. A. and Myers, B. A., "Sketching Intefaces: Toward More Human Interface Design", in *IEEE Computer*, 34(3), March 2001, pp. 56-64.
10. Lasseter, J., "Principles of Traditional Animation Applied to 3D Computer Animation", *Proceedings of the 1987 SIGGRAPH Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1987).*
11. *libpng*, Web page: http://www.libpng.org.
12. Lin, J., Newman, M. W., Hong, J. I., and Landay, J. A., "DENIM: Finding a Tighter Fit Between Tools and Practice for Web Site Design", *CHI Letters: Proceedings of the 2000 SIGCHI Conference on Human Factors in Computing Systems (CHI 2000).*
13. Lin, J., Thomsen, M., and Landay, J. A., "A Visual Language for Sketching Large and Complex Interactive Designs", to appear in *CHI Letters: Proceedings of the 2002 SIGCHI Conference on Human Factors in Computing Systems (CHI 2002).*
14. Matas, J., Galambos, C., Kittler, J., "Robust Detection of Lines using Progressive Probabilistic Hough Transform", in *Computer Vision and Image Understanding*, 78(1), April 2000, pp. 119-137.
15. *Open Source Computer Vision Library*, Web Page: http://www.intel.com/research/mrl/research/openv/.
16. Pierce, J. S., and Pausch, R., "Specifying Interaction Surfaces Using Interaction Maps", Technical Report, CMU-CS-01-100, January 2001.
17. Redström, J., Skog, T., and Hallnäs, L., "Informative Art: Using Amplified Artworks as Information Displays", *Proceedings of Designing Augmented Reality Environments (DARE 2000).*
18. *Textract*, Web page: http://www.structurise.com/textract/.
19. *zlib*, Web page: http://www.zlib.org.