Case Study: MAT

Chapter 10

Squeak: Object-Oriented Design with Multimedia Applications
Story

- MAT is a very simple WYSIWYG HTML-generating editor (by Aibek Musaev)
  - Simple, yet it supports sound, which most HTML editors don’t
- Only two classes, less than 10 methods
  - Great example of massive reuse
- Object Design critique: But should probably have touched more classes
- Interface Design critique: But complexities of Morphic (via reuse) become complexities of MAT
Using MAT

MAT new
openInWorld

- Use text commands to change style, color, and even hyperlinks (to URLs)
- Drag and drop to add SketchMorphs and SoundTiles
  - SoundTiles get mapped to a textual form to avoid changing sounds, which normal SoundTiles allow.

**SOUND|sound2**

This is a test
Generating HTML

- Red-halo menu to *Save as HTML*
  - Prompts for names for the GIF and AIFF files that it generates

- How it looks in Netscape:
Generated HTML

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
<title>Page created in MAT</title>
</head>
<body>
<pre><i><font size="7" color="#FF0000">This</font></i><font size="3"> </font><b><font size="5">a</font></b><font size="3"> </font><font size="3"><img src="mat2.gif"></font>
</pre>
<a href="sound2.aif">sound2.aif</a>
</body>
</html>
MAT’s Object Design

- Class MAT provides UI
- Modifications to ParagraphEditor does the work of generating HTML
Alternative Designs

Do it all in MAT?

- That’s not how the text classes work in Squeak
- PluggableTextMorph (MAT’s superclass) just deals with text *presentation*. Other classes take care of text *processing* (like editing).

Subclass ParagraphEditor to create MATParagraphEditor?

- Lots of classes to change
- We’re only *adding* functionality to ParagraphEditor — pretty safe

Change more classes?

- Valid: MAT’s methods frequently use *kindOf:* rather than delegate responsibility.
Walking through MAT

Adding drag-and-drop is easy

```plaintext
wantsDroppedMorph: aMorph event: evt
    " allow only SketchMorph, SoundTile, and TextMorph to be dropped "
    ^ (aMorph isKindOf: SketchMorph) | (aMorph isKindOf: SoundTile) | (aMorph isKindOf: TextMorph)
```
Accepting a Dropped Morph

Accepting a Dropped Morph is a little trickier

- If a SketchMorph, grab the Form and use a TextAnchor (a subclass of TextAttribute) to insert the Form into the text
- If a SoundTile, compose an appropriate string and insert it
- If another text, insert the content text

Delete the original morph to simulate it being “dropped in”
acceptDroppingMorph:event:

acceptDroppingMorph: aMorph event: evt
| contents form soundName txt attrib |
(aMorph isKindOf: SketchMorph) ifTrue: [
  form ← aMorph form copy. "Get the morph's form and make a copy"
  aMorph delete. "Get rid of the dropped morph"
  attrib ← TextAnchor new anchoredMorph: form. "Make a text 'attribute' out of the Form"
  txt ← '   ' asText. "Just some blank space"
  txt addAttribute: attrib from: 2 to: 2. "Make the Form the attribute of the blank space"
  self handleEdit: [textMorph editor zapSelectionWith: txt]. "Ask the editor to insert it" ].
acceptDroppingMorph:event:
(contd)

(aMorph isKindOf: SoundTile) ifTrue: [
    "generate corresponding link"
    soundName ← aMorph literal copy. "Make a copy of the sound NAME"
    aMorph delete. "Dump the dropped Morph"
    self handleEdit: [textMorph editor zapSelectionWith: "**SOUND|",soundName,'**']. "Insert"
].

(aMorph isKindOf: TextMorph) ifTrue: [
    contents ← aMorph contents copy. "Make a copy of the text"
    aMorph delete. "Dump the dropped Morph"
    self handleEdit: [textMorph editor zapSelectionWith: contents]. "Insert the text"
].
Adding “Save as HTML” to menu

```smalltalk
addCustomMenuItems: aCustomMenu hand: aHandMorph
    super addCustomMenuItems: aCustomMenu
        hand: aHandMorph.
    aCustomMenu addLine.
    aCustomMenu add: 'save as HTML' action: #saveAsHTML

saveAsHTML
    self handleEdit: [textMorph editor saveAsHTML]
```
ParagraphEditor

- Where all the real work goes on

- `saveAsHTML` deals with starting and finding sound links

- `linkFor`: does the actual processing of the sound

- `translateSqueakToHTMLtags`: handles all text styles and images
**saveAsHTML**

```smalltalk
saveAsHTML
  | sourceStream targetStream aLine start end specialCharacter text htmlFileName file link |

" ask the user for a name to create a new file "
htmlFileName ← FillInTheBlank request: 'Save as HTML'
  initialAnswer: 'mat.htm'.
htmlFileName isEmpty ifTrue: [^nil].
file ← FileStream newFileNamed: htmlFileName.
file text. "it'll be text, not binary"
specialCharacter ¨ $*. "Special character for Sound (and possibly future) tags"
```
"translate Squeak-rendered text to HTML tags"
text ← self text.
text ← self translateSqueakToHTMLtags: text. "Handle size, style, and color"
sourceStream ←ReadStream on: text.
targetStream ←WriteStream on: ".
[sourceStream atEnd] whileFalse: [
aLine ← sourceStream upTo: (Character cr).
"Now, look for links"
start ← 1.
[(start ← aLine indexOfSubCollection: (specialCharacter asString)
    startingAt: start ifAbsent: [0]) \= 0
    and: [start < aLine size]] "If there's a specialCharacter there..."
saveAsHTML (contd)

whileTrue: ["extract **LINK** combination"
  (aLine at: start+1) = specialCharacter
  ifFalse: [start ← start + 1]
  ifTrue: [
    (end ← aLine indexOfSubCollection: (specialCharacter asString)
      startingAt: (start+2) ifAbsent: [0]) ~= 0
    ifFalse: [start ← start + 2 "Eat up the second specialCharacter"]
    ifTrue: [
      (aLine at: end+1) = specialCharacter
      ifFalse: [start ← end+1]
      ifTrue: [
        "Grab the link string from aLine, convert via linkFor:"
        link ← self linkFor: (aLine copyFrom: start+2 to: end-1).]}.]
saveAsHTML (ended)

"Now insert the link into the output line"

    link isNil ifFalse: [
        aLine ← aLine copyReplaceFrom: start to: end+1 with: link
        aString.
        ]. start ← end + 2.
        ].
        ].
        ]. " whileTrue: "
        targetStream nextPutAll: aLine; cr. "Put aLine into the output HTML file"
    ]. " whileFalse: "
    " put HTML data into file "
    file nextPutAll: targetStream contents.
    file close.
**linkFor:**

Converts the SampledSound from the library with the given name into an AIFF file to link in

```
linkFor: aString
  | fileName returnStr soundName entry samples samplingRate f |
  (soundName ← self isStringASound: aString) ifNotNil: [  
    "SOUND link is encountered"
    fileName ← FillInTheBlank request: 'Save as AIFF' initialAnswer:  
    soundName,'.aif'.
    fileName isEmpty ifTrue: [^nil].
```

9/22/00 Copyright 2000, Mark Guzdial
linkFor: (contd)

" retrieve samples and sampling rate info "
entry ← SampledSound soundLibrary
   at: soundName asString
   ifAbsent:
      [self inform: soundName asString, ' not found in the Sound Library'.
         ^ nil].
"Convert the SampledSound representation into AIFF"
entry ifNil: [^ nil].
samples ← entry at: 1. "Get the actual samples"
samples class isBytes ifTrue: [samples ← SampledSound
   convert8bitSignedTo16Bit: samples]. "Is this needed?"
samplingRate ← (entry at: 2) asInteger. "Get the sample rate"
linkFor: (end)

f ← (FileStream fileNamed: fileName) binary.
AbstractSound new storeAIFFSamples: samples samplingRate:
samplingRate on: f.
f close.
returnStr ← '<A HREF="',fileName,'">',fileName,'</A>'.
^returnStr.
]

" don't know what to do, return nil "
^nil.
translatesqueakToHTMLtags:

- Most interesting method
- Deals with
  - TextEmphasis (e.g., italics)
  - TextURL (link to a URL)
  - TextColor
  - TextFontChange
  - TextAnchor (for images)
translateSqueakToHTMLtags:

translateSqueakToHTMLtags: aText

" translate Squeak-rendered text to HTML tags. TextAttribute's considered: TextEmphasis, TextURL, TextColor, TextFontChange "

| runArray runs value values readStream targetStream rgbColor fontSize fileName |

readStream ← ReadStream on: (aText string). "For reading the actual text"

targetStream ← WriteStream on: ''. "For writing the actual text"

"Store a default page header and default title into the target"

targetStream nextPutAll: (HTMLformatter startPage: 'Page created in MAT').

targetStream nextPutAll: '<PRE>'. "All text should be considered pre-formatted"
Translation (contd.)

"The 'runs' are where the text attributes apply"

runArray ← aText runs. "Returns a RunArray"

runs ← runArray runs.

values ← runArray values. "Gets the attributes"

1 to: (runs size) do: [:index |

value ← values at: index. "Get the attributes associated with this run"

value do: [:attr |

(attr isKindOf: TextEmphasis) ifTrue: [

(attr = TextEmphasis bold) ifTrue: [targetStream nextPutAll: '<B>'].

(attr = TextEmphasis italic) ifTrue: [targetStream nextPutAll: '<I>'].

(attr = TextEmphasis underlined) ifTrue: [targetStream nextPutAll: '<U>']. ].].
Translation (contd.)

(attr isKindOf: TextColor) ifTrue: [
    rgbColor ← MAT generateHTMLRGBfromColor: (attr color).
    targetStream nextPutAll: '<FONT COLOR="',rgbColor,'">'.
]

(attr isKindOf: TextFontChange) ifTrue: [
    fontSize ← attr fontNumber + 2.
    targetStream nextPutAll: '<FONT SIZE="',fontSize asString,'">'.
].

(attr isKindOf: TextURL) ifTrue: [
    targetStream nextPutAll: '<A HREF="',attr info,'">'.
].
"Deal with the image"
(attr isKindOf: TextAnchor) ifTrue: [
    fileName ← FillInTheBlank request: 'Save as GIF' initialAnswer: 'mat.gif'.
    fileName isEmpty ifFalse: [
        GIFReadWriter putForm: attr anchoredMorph onFileNamed: fileName.
        targetStream nextPutAll: '<IMG SRC="',fileName,'">'.
        " skip '   ' part ".
    ].
].
"Put the actual string out onto the target"

targetStream nextPutAll: (readStream next: (runs at: index)).

"Did the run just end? Put in the ending tag!"

value do: [:attr |

(attr isKindOf: TextEmphasis) ifTrue: [

(attr = TextEmphasis bold) ifTrue: [targetStream nextPutAll: '</B>'].

(attr = TextEmphasis italic) ifTrue: [targetStream nextPutAll: '</I>'].

(attr = TextEmphasis underlined) ifTrue: [targetStream nextPutAll: '</U>']. ].
Translation (contd.)

(Attr isKindOf: TextColor) ifTrue: [targetStream nextPutAll: '
</FONT>'].

(Attr isKindOf: TextFontChange) ifTrue: [targetStream nextPutAll: '
</FONT>'].

(Attr isKindOf: TextURL) ifTrue: [targetStream nextPutAll: '</A>'].

].

].

targetStream nextPutAll: '</PRE>'.
targetStream nextPutAll: (HTMLformatter endPage).
^targetStream contents.
User evaluation of MAT

- Evaluated MAT as an annotation tool for students’ use of MuSwiki
  - MuSwiki is a Swiki (as in PWS) but without HTML or a normal browser
  - Users can put any Morph into any page
  - Example use: Students doing collaborative CRC Card analysis with MAT annotations
Example Use of MAT
MAT Usability Survey Results

<table>
<thead>
<tr>
<th>Statement</th>
<th>Average (standard deviation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>It is easy to add text to a MuSwiki page using a MAT.</td>
<td>3.2 (1.2)</td>
</tr>
<tr>
<td>It is easy to modify text that is in a MAT</td>
<td>3.6 (1.1)</td>
</tr>
<tr>
<td>It is easy to change the color of text in a MAT</td>
<td>3.3 (1.0)</td>
</tr>
<tr>
<td>It is easy to resize a MAT window</td>
<td>2.0 (0.9)</td>
</tr>
</tbody>
</table>
Summary of Design and Usability Evaluation

- Design is minimal, lots of reuse
  - But tests objects, as opposed to delegating responsibility to objects (which can be later over-ridden for extensability)

- Usability is limited by reuse
  - Yes, it works, but things like resizing are just as hard to do to MAT as to general Morphs