Case Study: Prototyping a Play-Writing Workbench

Chapter 11

Squeak: Object-Oriented Design with Multimedia Applications
Story

- Prototyping: Build “enough”
  - Demonstrate the key features
- Learner-Centered Design requires focus on support and motivation, as well as usability
- Play-Writing Workbench Prototype through two iterations
- Usability suggests that the goal design should be usable, but points out problems that may not have been noted otherwise
Goal: A Play-Writing Workbench

A toy for an eight-year-old, to encourage reading/writing and interest in theatre
Desired Features

- Play construction via drag-and-drop
  - Actors and lines as separate graphical objects
- Actors as heads that speak lines
  - Ability to tailor the look of the actor and sound of the voice
- Share plays via a Swiki
Scenarios

- User should be able to define a script
  - Specifying actor and voice characteristics
  - Specifying order of actors and their lines
- User should be able to “play” the script
  - Actors speak their lines, in order
### A ScriptingBench CRC Card

<table>
<thead>
<tr>
<th>Scripting Bench (not stage)</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>Actor, Script, BenchCard</td>
</tr>
<tr>
<td>sets up speakers, script, controls</td>
<td></td>
</tr>
<tr>
<td><em>(Starts playback of play)</em></td>
<td>Script</td>
</tr>
<tr>
<td>allows sharing of play</td>
<td>swiki Action</td>
</tr>
<tr>
<td>allows loading of play</td>
<td>swiki Action</td>
</tr>
</tbody>
</table>
An Actor CRC Card

<table>
<thead>
<tr>
<th>Actor</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

- B: Makes appearance of actor
- C: Speaks lines
  - Allows for editing of voice characteristics
  - Speaker Morph
  - Speaker
  - Speaker
Script/ActorLines CRC Cards

- Scripts know the lines in order
- ActorLines know what’s to be said and who says it
Opt *not* to do UML class design

- Instead, decide to play with Speaker and GesturalVoice, first.
  - Didn’t know the technology, wasn’t sure how it was going to connect
- Not a bad strategy for a prototype
  - But beware the prototype that becomes the final product...
Implementing a PlayBench

Subclass off AlignmentMorph, as do other Morphic applications

AlignmentMorph subclass: #PlayBench

instanceVariableNames: 'script scriptBench actors '

classVariableNames: "

poolDictionaries: "

category: 'PlaySpace'
Creating the PlayBench

- Create based on newColumn
  new
    | me |
    me ← super new "Accept submorphs vertically"
      orientation: #vertical;
      hResizing: #spaceFill;
      vResizing: #spaceFill.
    ^me setUp "Set up the pieces"
Setting up the PlayBench

Use the pieces identified in the CRC Cards

**setUp**

"Add in the pieces"

self addMorph: (self addScriptingControls).

self addMorph: (self addScriptingBench).

self addMorph: (self addActors).

^self
How much to do in a prototype?

- Should I create classes for ScriptingControls, ScriptingBench, Actors?
  - Absolutely! Will make it easy to extend later
  - Design flaw: I didn’t. Just used pluggable pieces

- “How much do I implement now according to the requirements (when I’m not sure that the goal is right), and how much do I put together more simply?”
  - Do enough to demonstrate the main functionality or purpose.
Implementing ScriptingBench

ScriptingBench becomes a PluggableTextMorph

(Make it bigger so that it looks different than the actors’ names’ text areas.)

addScriptingBench

"To start, the scriptingBench is just a simple PluggableTextMorph"

scriptBench ← PluggableTextMorph on: self text: #script accept: #script:. 

scriptBench extent: (scriptBench extent * 2.0). "Make it larger"

^(AlignmentMorph newColumn addMorph: scriptBench)
Providing a default script

```plaintext
script: someText
  script ← someText.
  ^ true "To accept the accept"

script
  script ifNil: [ script ←
'Bert: Hi, Ernie!
Ernie: Hello, Bert. How are you?
Bert: Oh, I'm fine, Ernie. Where are you going?
Ernie: Where else? I am going to Sesame Street!'].
  ^script
```
ScriptingControls =>
Single Play Button

addScriptingControls

"For now, just a play button"

^PluggableButtonMorph on: self getState: nil action: #play label: #playLabel.

playLabel

^'Play'
Implementing Actors

We will create PlayActor objects

A name, a voice (to delegate actually “speaking”), a face (a Morphic representation), and a “nameSlot” to deal with young users...

Object subclass: #PlayActor

instanceVariableNames: 'name voice face nameSlot'
classVariableNames: "
poolDictionaries: "
category: 'PlaySpace'
Adding actors to the PlayBench

```smalltalk
addActors

| actorRow |
actorRow ← AlignmentMorph newRow.
"Let's start with just two actors"
actors ← OrderedCollection new.
actors add: (PlayActor named: 'Bert').
actors first voice: (Speaker man).
actors add: (PlayActor named: 'Ernie').
actors last voice: (Speaker child).
"Ask each actor for its morphic representation, and store in the row"
actors do: [:a | actorRow addMorph: (a morph)].
^actorRow
```
Assigning a Morphic Representation

For now, the representation will just be an actors’ name.

morph

"Start out with just a textMorph for name"

nameSlot ← PluggableTextMorph on: self
text: #name accept: #name:. 

^nameSlot
Expecting bugs...

When you’re prototyping, *expect* lots of bugs.

- Make debugging easier for yourself!
- For example, make printString provide useful information

```smalltalk
printOn: aStream

aStream nextPutAll: 'a PlayActor named: ', name.
```
Implementing Play

*A child types in actors’ names, and clicks “Play” – probably won’t “Accept” the names.*

That’s why we have to have the **nameSlot**: To accept *for* the user, if Play is selected.

```
play

"Play each of the lines in the scriptingBench"

| scriptStream scriptLine actor line thisActor |

"Make sure that we have the script, and that it's a string"

scriptBench accept.

script ← scriptBench getText asString. "Get the script"

"Make sure all the names are saved"

actors do: [:a | a nameSlot accept].
```
Play (contd.)

```smalltalk
scriptStream ← ReadStream on: script.
[scriptStream atEnd] whileFalse:
    [scriptLine ← scriptStream nextLine.
      actor ← (scriptLine copyUpTo: $:) copyWithout: (Character space).
      Transcript show: actor,' is going to say "',line,""); cr.
      thisActor ← (actors detect: [:a | a name = actor]).
      thisActor say: line.
      SoundPlayer waitUntilDonePlaying]
```
Speaking is the easiest!

- It’s just delegation!
  
  say: something
  
  voice say: something.
Actually, there’s a trick

In the first version of play, SoundPlayer waitUntilDonePlaying wasn’t there.

Actors started speaking in the middle of the others’ lines.

But there was no built-in way to wait until the end of voices

So I looked for a way to wait until the end of all sounds
How to waitUntilDonePlaying

I found that `waitUntilDonePlaying: aSound` checked `ActiveSounds`

I found that `ActiveSounds` was empty when all sounds were done playing.

`waitUntilDonePlaying`

"Wait until any sound is no longer playing."

[PlayerSemaphore critical: [ActiveSounds isEmpty not]]

whileTrue: [(Delay forMilliseconds: 100) wait].
First Prototype Iteration

Bert: Hi, Ernie!
Ernie: Hello, Bert. How are you?
Bert: Oh, I'm fine, Ernie. Where are you going?
Ernie: Where else? I am going to Sesame Street!
Choosing to go on...

- Is this enough?
  - Will this be motivating enough for an eight-year-old learner?
  - It does talk, but the *faces* would make it more appealing.

- Added to **addActors**
  
  "Ask each actor for its morphic representation, and store in the row"

  actors do: [:a | actorRow addMorph: (a morphWithFace)].

  ^actorRow
Creating morphWithFace

It’s actually hard to do right from Speaker, so had to replicate some code.

Should have improved Speaker instead.

``` Smalltalk
morphWithFace
| faceColumn |

faceColumn ← AlignmentMorph newColumn.

"Assume that the voice has been set with a Speaker by this point"
voice isNil ifTrue: [^self error: 'Must set voice first'].
voice makeGestural.
```
morphWithFace (contd.)

face ← HeadMorph new. "Generates a random head"
(voice findAVoice: GesturalVoice) head: face. "Connect voice to head"
faceColumn addMorph: face.
"Start out with just a textMorph for name"
nameSlot ← PluggableTextMorph on: self text: #name accept: #name:.
faceColumn addMorph: nameSlot.
^faceColumn
Faces aren’t moving!

The lips aren’t synching with the voices

waitUntilDonePlaying must be blocking the other process or step method

waitUntilDonePlaying

"Wait until any sound is no longer playing."

[PlayerSemaphore critical: [ActiveSounds isEmpty not]]

whileTrue: [(Delay forMilliseconds: 100) wait.

Processor yield. World doOneCycle].
Completed Prototype

Ernie

Bert

Bert: Hi, Ernie!
Ernie: Hello, Bert. How are you?
Bert: Oh, I’m fine, Ernie. Where are you going?
Ernie: Where else? I am going to Sesame Street!

Play
Evaluation of the PlayBench

- First round: Expert Review
  - Ask experts in children’s software what they thought:
    - “Will the voices be clear enough?”
    - “Will they be able to tell what the faces are doing?”
Second Round: Observation

- Showed my eight-year-old the PlayBench
  - “Cool! It’s *so* awesome!”
  - Changed an actor’s name to two words—which broke the space elimination in the script.
  - “Can I change how they look?”
  - “Can you write that (name) down for me? Just the way I wrote it!” Typing of actor’s names was tedious.
  - Didn’t know how to get to another line!
  - “You can’t make his voice sound very good.”
  - Didn’t pronounce as he expected
Summary

- Prototyping doesn’t mean you give up on design
  - But you emphasize key functionality
  - For learners, can’t skip out on support and motivation
- PlayBench should have been a little better designed, e.g., classes, not just pluggables
- Evaluation suggests that the final version would have right features
  - But some lessons (like voice issues) are being learned now, at the prototype stage.