Implementing DBJr with Worlds

Ted Kaehler, Alex Warth and Yoshiki Ohshima

This material is based upon work supported in part by the National Science Foundation under Grant No. 0639876. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

VPRI Memo M-2009-015

Implementing DBJr with Worlds

Ted Kaehler, Alex Warth and Yoshiki Ohshima

(clarification of our email of 6 July 2009 and demo on 14 July 2009)

Introduction

How can we specify the behavior and semantics of DynaBook Junior (a HyperCard-like stack of pages) in a minimal way? This problem is interesting because DBJr is an application, and it happens to be an application that is difficult to specify using normal class-instance conventions.

In Hypercard, a stack is a collection of pages (or cards). Many pages may have the same structure (e.g., what fields exist, and where these fields are placed) but hold different data. This is accomplished by having a background that is shared by those pages. When the user flips through the pages, she always sees the same objects — the fields of the background — which are filled in with the data that is stored in the current page. So in standard object-oriented parlance, the background is like a class, and a page is like an instance.

Any page may have "page-specific" objects (e.g., pictures, text fields, annotations) that are only shown on that page. The data structures that hold these objects are particularly complex: usually, the background class will have a subclass that has additional instance variables to hold the page-specific objects.

Objects that are part of a background cannot be stored in the instance variables of the page, because they would be in every page-instance. They must be stored using some other mechanism such as class variables. And both shared and page-specific variables must be able to be added and deleted dynamically, because the user may add and remove fields and pictures at any time.

Not only is this specification of Hypercard complex, but the code that implements it is complex too. (See diagrams below.) Our proposal is to get the same functionality while minimizing this complexity. We do this using Worlds.

Using Worlds to Implement Hypercard-like Semantics

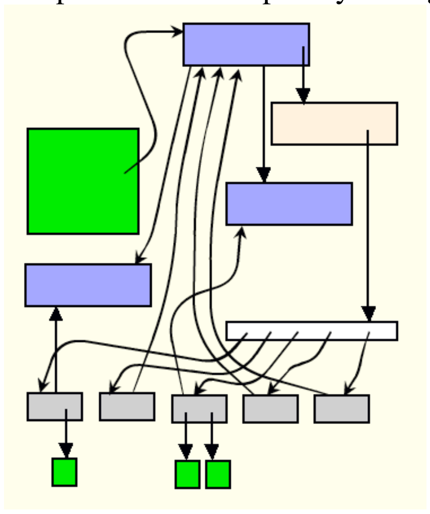
We start with an ordinary rectangle with text fields in it. The concept of a background is easy to implement: it is just a World that views the rectangle. Pages that share the same background will all be Worlds that are sprouted from that "background World". In this implementation scheme, a stack is an ordered collection of Worlds, each of which represents a page.

An object that exists on a background may be viewed and modified through several different Worlds. Thus in each World, the object may have different contents. In fact, all of the properties of the object, e.g., its foreground color and position, may be modified in a page-specific way using this mechanism.

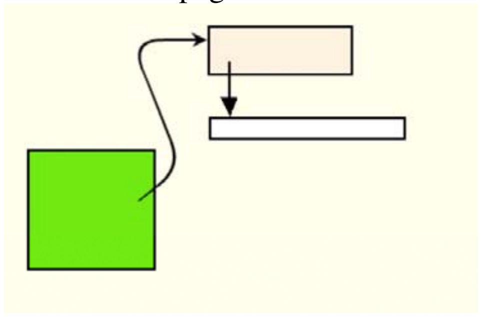
The following operations are enough to allow the functionality of DBJr:

_. **To display a page**, we find its top LBox (display object), and invoke the method that draws it on the screen. This must be done inside the World that represents this page - that way all of the objects will be seen as they are in that World.
_. **To add a new page with background B**, we sprout a new World that is a child of B's World and add it to the stack's ordered collection of pages. Whenever an object is modified in this new page, only the "delta" of the object (i.e., the difference between this page's version and the background's version) is stored in the current World. Objects that are not modified are simply inherited from the background. This mechanism minimizes the memory overhead of Hypercard-style sharing, and is provided entirely by our Worlds implementation; the application's programmer does not have to write any additional code in order to support it.
_. **To add a new field / object to (or modify an object in) a page P,** we simply evaluate the code that does so in the World that represents P. That way the changes will only be visible in P (and not in the background or the other pages).

Here are two figures to emphasize the complexity savings:



Data structures needed for DynaBook Junior (HyperCard). The actual object structure is more complex than this. Green are display objects, with the large one being the page. Blue are backgrounds, tan is the stack, white is the collection of pages. Gray are the data structures for individual pages.



The same functionality in DynaBook Junior using Worlds. The white collection holds a World for each "page".

Our Prototype Implementation and Demo

We have built a demo of DBJr constructed from Worlds.

In the stack, each page is a separate World. The stack really has only one page, which is viewed through different Worlds. When a page is turned, its objects and their contents change simply because the page is viewed from a different World. This includes changing the text, adding and removing objects, and changing details of objects. Any page can be a background for any other page. Multiple backgrounds are no problem.

We used the LObject system and Ted's LWordWrapLayout text fields.

The 152 kb of source code for the old OUStacks is now reduced to seven small methods in class LStack.

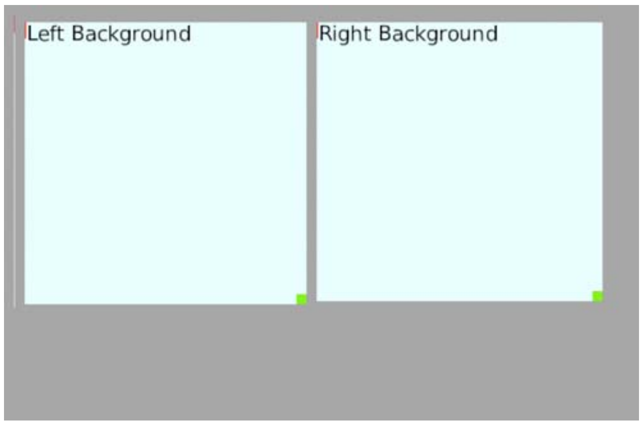
Alex's Squeak-based Worlds implementation is also remarkably small. It has three classes with 46 methods, most of which are only one line long.

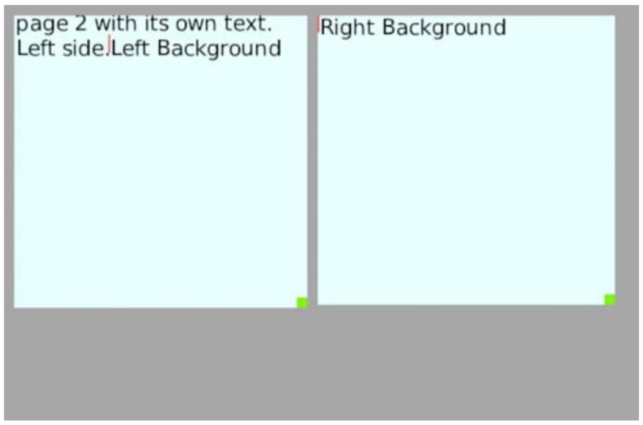
We will need to do a reformulation of this as we search for a solution to the "Focus Problem" in Worlds. When a user has a stack open that has an audio track, and she changes the volume, does that change belong in the World of that page, or in the OS of the computer? Every single action that the user takes needs a policy that says whether it is in the page or outside of it. There may also be actions by system processes that are not clearly in the page's World or out of it. If a change of display shrinks the page's window, is the size change part of that page? These problems are made worse by the very general way that Worlds trap changes.

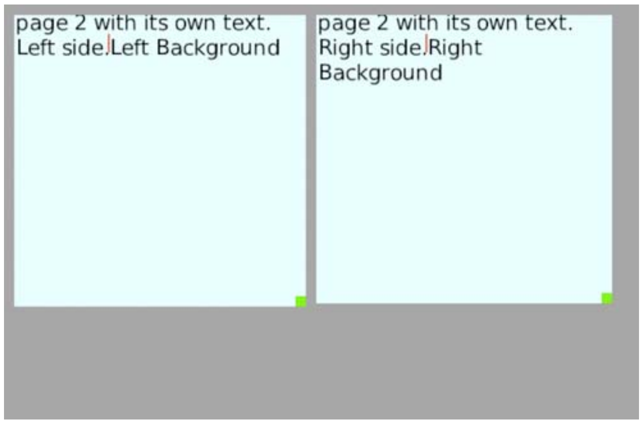
At the moment, though, we have 'broken through', and we have a demo.

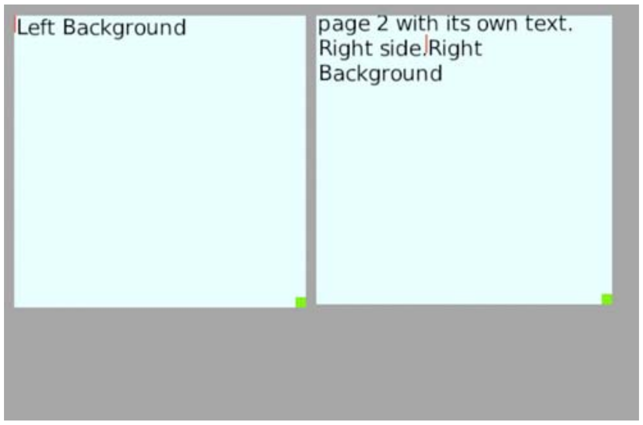
In this demo, there are two stacks. It is important that we show two independent stacks to demonstrate that we are putting views through several worlds in the same screen. In this sequence of screen shots, we turn to page 2 in the left stack, then to page 2 in the right stack. Then we turn back to page 1 in each stack. Note that fullDrawOn is using three different worlds as it renders different parts of the screen (the top world, a page of the left stack, and a page of the right stack).

We are so happy to get this working.






page 2 with its own text.
Left side.Left Background
page 2 with its own text.
Right side.Right Background



Try It Yourself!

The following steps will recreate our demo. (Important: this only works in our "Moshi" Squeak image. Bring in Worlds2-aw.cs, WWorld-A-tk.1.cs, WWorld-B-tk.4.cs, Worlds-Morph-A-tk.5.cs, Worlds-DBJr-B-tk.1.cs, then look at file 'LStack WWorld workspace') These instructions are here so that we won't lose them. This demo was difficult to get working.

1. Create a graphical object on the screen. Choose "Be a Page in a New Stack". This creates an array of Worlds, forces the current object to be a World, and stores it as the first element of the pages array.

2. Embed other objects in the object created in (1). Modify those objects in any way. Also embed two text fields.

3. Choose "Sprout a World". You are now looking at page 2, which looks exactly like page 1. Replace the text in the fields with new text.

4. Choose "Sprout a World" again, and choose page 1 as the background. Change the text in the fields again.

5. Use Cmd-Right-Arrow to flip between the three cards.

6. Go to page 1 and move a field. Page 2 inherits from page 1. Go to page 2 and see that the field has moved. If you move the field on page 2, you will see that the field will not move in its sister pages or in the background.

Extras

Suppose you have a Z-ordering of objects, some created on this page, and some from the background. Turn to the background page. If a new object is added to the background page at a certain Z-depth, what happens when you return to the daughter page? There is a strategy for merging the new object into the existing mixture of background and specific object on the daughter page. (This algorithm exists in the current DBJr. It now becomes a general property of Worlds that are sprouted from each other.)

Another way of saying this is that certain special objects are a mixture of shared and not shared. The contents collection of a page is one of these. It is a mixture of objects in the background world and objects in the current world.

There needs to be a way to store the stack (array of worlds) world in a file. The current SmartReferenceStream and ImageSegment code should be able to handle this.

Because Worlds can inherit from each other, sharing is maximal. Everything that can be shared between similar pages is shared automatically.