Alternative Technologies-Ruby on Rails

Pierce T. Wetter III

Myth: Rails is a better framework for Ajax than WO.

Reality: WebObjects is actually a better framework for use with Ajax libraries than Rails because it has a better component system than Rails. You spend a lot of time coding little tiny XML and HTML generators when doing Ajax and WO's component system makes that very DRY (Don't Repeat Yourself).

Myth: WebObjects is hard to use with Ajax.

Reality: WebObjects is easy to use with Ajax, it's just that there is only one known library for Ajax-WO support, and its not well documented. Even then the library only goes so far in that it just provides new components to wrap a few script.aculo.us tags. I think its more of a documentation gap, not a code gap.

Additionally, the WO documentation has always pushed people erroneously towards component actions and away from direct actions. If you use 90% direct actions like I do in my application, (See my "WebObjects on Rails" post), you'll find that direct actions are simple to code in reality, and very much like Rails which basically doesn't have component actions at all.

Which is to say, if you avoid much of the power of WO, and don't use component actions, ajax will be easier. If you do use component actions - and I have yet to work on a project that doesn't - then ajax use seems like it will blow your page cache, as described below. So it really is easy to use AJAX in WO. Really easy. It's just that it doesn't work.

[mschrag: While I agree with the above commenter that component actions provide a huge amount of power in WO, it is NOT true that Ajax and component actions are incompatible. Project Wonder's Ajax components directly addresses the use of component actions with Ajax in a way that does NOT blow the page cache. While it is true that the implementation of these capabilities inside of Wonder was non-trivial, it demonstrates that it is, in fact, possible, and if you use the PW components along with ERXSession, you will get this capability for free.]

Myth: You can't assign the name or id value of tags in WebObjects.

I'm mentioning this one explicitly, because I had someone ask me about this recently, because they were trying to assign a WYSIWYG JavaScript editor to a text area.

Reality: Actually, what really happens is that if you DON'T have a name=value or id=value line for your TextArea in WebObjects, it will generate a unique one for you. But you're more than welcome to specify one in your .wod file, and WO will use that. It's then up to *you* to make sure its unique, so that you don't have multiple text area tags with the same name. In other words if you only have one textarea you wish to attach a Javascript WYSIWYG editor to, just add:

```
textarea: WOText
{
  id='wysiwyg';
  name='wysiwyg';
  value=textValue;
}
```

in your .wod file and you can use id with your Javascript just fine.

Myth: WebObjects generates HTML

Reality: Just because the components of your .wo file are .html and .wod doesn't mean WebObjects can only generate HTML. WebObjects is really at heart, a very, very, very, complicated printf. The result of a component can be HTML, XML, JavaScript, or even binary data. You can actually put whatever you want in the .HTML and treat WebObjects like a giant merge engine. For instance, you could generate PDF files using WebObjects; they're just text, and you could substitute text into the middle of the PDF boilerplate pretty easily.

Background

We haven't used a lot of JavaScript at www.marketocracy.com, for a very simple reason: We don't have the testing staff to fire up 10 different browser variations to test the site. And lets face it, programming in JavaScript sucks.

Since someone invented that cool acronym though, that's changing. You would think it shouldn't matter that something has a name, but it does. I remember when the GoF Design Patterns book came out. There was nothing in there I hadn't figured out on my own, but now they had a name! I could say "Singleton" to my co-workers and they would know what I was talking about. I could buy a junior engineer the book, and he would soon be programming at a much higher level.

So having a name helped, and the result is that there are a lot of cool toolkits out there. Since someone else wrote the JavaScript, that means I don't need to test 10 different browser variations.

Which means I can make the site much easier to use and more interactive. Huzzah!

Research

So the first thing I did was that I'd heard that Rails was cool and made Ajax easy. I'd heard that before, but that was when the version number was 0.13... So I went out and bought the Rails books and you know what I found?

It's not that Rails rocks with Ajax, its that the Prototype Javascript library rocks. The documentation makes a big deal about doing Ajax with one line of code.

```
<div id='<= posting.id'>
  <%= link_to_remote  [div to update] [link options] -> %>
</div>
```

The reality is that all Rails is doing is writing one line of JavaScript. From the Ajaxy "rating" code I've been adding to my app, look at the following:

```
<div id='<WEBOBJECT name=postingID></WEBBOBJECT>'>
  <a href="#" onclick="new Ajax.Updater('<WEBOBJECT name=postingID></WEBBOBJECT>',
       '<WEBOBJECT name=ratePosting></WEBOBJECT>', {asynchronous:true, evalScripts:true}); return false;">1</a>
</div>
```

Ok, so lets break it down. The way Ajax.Updater works from the Prototype library is you give it the id of a DOM object and a URL, and it replaces the DOM object with that id with the contents of the URL. Usually, you specify the area to update with a div tag which I've shown for completeness. For ratings I would need 1 div tag to enclose all 5 rating stars: <a><a><a><a><a> . I would also use a WOGenericContainer to generate the div tag with the right id rather than using the id='<WEBOBJECT name=postingID></WEBBOBJECT>' line, but I wanted it to be obvious it was div tag.

So the Ajaxy part is the <a> tag, not the div tag.

Here I have a manually built <a> tag with an onclick bit of javascript. That makes a single call out to the Prototype library, which has this cool call:

```
new Ajax.Updater( elementid, url, options)
```

It does something very simple: It pulls the HTML from the specified URL, and replaces the element with the specified id with the downloaded HTML.

The two WebObjects tags specify the element ID and the link, they're just a WOString and a WOActionURL:

```
postingID: WOString { currentPosting.primaryKeyString; }
ratePosting: WOActionURL { see discussion }
```

Now in my case, I'm a direct action snob. So my WOActionURL would look like the following:

```
ratePosting :WOActionURL
{
  directActionName="PostingRater";
  ?pkey=currentPosting.primaryKey;
  ?rating=cRating;
  ?wosid=NO;
}
```

This produces a result similar to Rails, because in rails you have to define an action for each class of link. In my case I tie directactions to pages /components, so my "PostingRater" page would return component-level HTML (minus any HEAD/BODY tags) that matched the existing definition. Since we're using WebObjects, that turns out to be trivially easy if we build the enclosing tag with a component PostingRater can just look like:

```
<WEBOBJECT name=RatingDiv></WEBOBJECT>
```

Using component actions, it could be even simpler:

```
ratePosting: WOActionURL { action=ratePosting;}
```

Because WebObjects, unlike Rails can have stateful components, the RatingDiv component could actually have all the logic and return self as the result of the action:

```
public WOReponse ratePosting
{
  currentPosting.ratePosting(currentRating);
  return self; // since this is called from JavaScript
               // return just myself, not self.page
               // this will tell WO to render only
               // this as a result.
}
```

That is, the link for the javascript would go into the RatingDiv component with everything already setup: the current posting, the current rating. Returning self then causes the div to regenerate.

However, there's a gotcha there, because component actions called from javascript count towards your backtracking count. Since someone might be going down through the page clicking rating after rating, that could be a problem. If your max backtracking count was 10, and you had 20 postings on a page, they wouldn't be able to rate the 11th page.

So where Rails has one line, I have two WOTags, but those could (and should) be easily combined by creating a WODynamicElement to generate the link directly. In essence, it would be pretty easy to create a "RemoteLink" WODynamicElement that did everything the Rails "link_to_remote" call did, namely take an id to update along with all the options WOActionURL takes.

But additionally the WebObjects solution is superior in a number of ways to the Rails solution. Rails has support for both "partial pages" and "components" but the reality is that components are pretty heavyweight/slow and partials don't quite do what you think. So the WebObjects solution, by wrapping both the logic and state into a component ends up being more DRY (Don't Repeat Yourself) and better encapsulated than the typical Rails solution. In the page where you want the Ajax bit, you specify the component, and you put the ajax handling logic in the component itself.

I have yet to see anything in the Rails Ajax support that couldn't be just as easily done in WO via a combination of WOComponents and WODynamicElements. Pretty much everything in the Rail "Ajax support" are one-liners in Prototype. In fact, because pages/components can have state, its probably *easier* to do it in WO than in other systems. Also, I think you could build a set of WOComponents that provided *superior* Ajax support than Rails. It's just that no one has written the supporting WODynamicElements or WOComponents.

Now looking at the dojo Hello World example Ahmet, I have to say, that there's nothing in WebObjects that precludes doing anything in this. 90% of this tutorial is JavaScript that isn't specific to any particular technology. The parts that are specific to PHP/ASP/ColdFusion are defining a new page just like you'd have to do in WebObjects. Just use a WOActionURL where it says url: in the Javascript and you're done.

I also think, that from this example, dojo is a bit weak as a Javascript toolkit. You'll spend a lot of time wiring stuff up in Javascript with Dojo compared to Prototype. Or contrast dojo with the Yahoo UI library:

http://developer.yahoo.com/yui/

The whole "Dialog" thing in yui is cool:

http://developer.yahoo.com/yui/container/dialog/index.html

And would lend itself quite readily to the model WO has of building pages out of pieces.

Then again, dojo is only at version 0.3. The WYSIWYG editor is nice, but I really don't want to have to define my dialog boxes in yet another tag language. HTML is good enough for me.

So exactly what problems are you having with dojo/wo? Consider that the problem may be dojo, not WO in particular. You could easily create a WODynamicElement for every new dojo tag, collect them in components, and have something much easier to use than straight dojo.