Development-Generating Static Pages

Building static sites from WebObjects

During WWDC last week, a member of the audience asked at a WebObjects session how to use WebObjects to create static pages from a dynamic site. Bill Bumgarner fielded the question, as I recall. This is a problem that comes up many times over and over when proposing any sort of dynamic site. At one time, a brief description of how to handle this was included in the Programming Topics of the WebObjects on-line documentation, written by the Apple employee who set up the initial web site for the BBC; that note did not survive the many purges of the web site - which is all to the good, as it was very brief and cursory.

There are many reasons to create static sites from a dynamic engine; the BBC chose to do so out of concerns for the performance of WebObjects. More reasonable justifications are to use html as a portable documentation format; for example, to create a corporate image manual that is distributed on CD. I have been involved with several projects that have done this, including the two mentioned above.

I will entirely skip over how to create dynamic pages from within WebObjects and concentrate on how to create static pages from within a page level component. Dynamic content is a simple problem to solve, and it is done many times over. The problem of creating static pages divides neatly into two: the obvious part is how to write the page to an html file, which is usually referred to as "spidering"; the less obvious part is that all the page links have to somehow be converted from WebObjects dynamic actions into static URLs.

Spidering

First, you need a mechanism to visit every page on your site. I will typically create an action in an administrative application for this purpose. I will have a database structure with a record (an enterprise object) for every page to be written out. This will contain relationships for the children of the page, and for the parent. The top level page can be identified because it has no parent; this structure can also be extended to represent multiple sites, or top level pages, within a single database if required. My page, or node, EO will have a method to write itself out; the file system location to write to will be calculated from the node path to the current node, using a file system name in the EO to label a path.

I use the following method within a component to write out the page, which is called from the spidering code; node is the EO that represents a page or node, and htmlPath is an obvious method.

```
public void writeToHtml() {
    WOResponse r = null;
    String c = null;
    try {
        File folder = new
File(NSPathUtilities.stringByDeletingLastPathComponent(node().htmlPath()));
        folder.mkdirs();
        BufferedWriter out = new BufferedWriter(new FileWriter(new
File(node().htmlPath())));
        r = generateResponse();
        if (r == null) NSLog.debug.appendln("writeHtml: no response");
          c = r.contentString();
        if (c == null) NSLog.debug.appendln("writeHtml: no content");
          out.write(c);
        out.flush();
    } catch (IOException e) {
        System.err.println(e.getMessage());
    } catch (Throwable e) {
        System.err.println("writeToHtml(" + node().htmlPath() + "): " +
e.getMessage());
    }
}
```

An example implementation for the htmlPath() method follows:

```
public String htmlPath() {
    return
NSPathUtilities.stringByAppendingPathComponent(PLUtilities.defaultForKey(n
ull,
        "docroot", "/Library/WebServer/Documents").toString(), urlPath());
}

public String urlPath() {
    String fileName = name() + primaryKeyString(this) + ".html";
    return "/" +
NSPathUtilities.stringByAppendingPathComponent(site().name(), fileName);
}

static public String primaryKeyString(EOEnterpriseObject object) {
    NSDictionary dict =
EOUtilities.primaryKeyForObject(object.editingContext(), object);
    String result = "";
    java.util.Enumeration enumerator = dict.objectEnumerator();
    while (enumerator.hasMoreElements()) {
        Object anObject = enumerator.nextElement();
        result += anObject.toString();
    }
    return result;
}
```

A more intelligent implementation may be appropriate, to build a path string depending on the node hierarchy, or to take a file name from EO attributes, etc.

Dynamic/Static Links

The only actions that can be used in components for this type of site are WOHyperlinks; these can be converted to static page references, whereas form actions, etc, cannot. The base assumption is that any pages will be served from a file system, not from a web server, and so cgi will not be supported. In theory, you could make use of Javascript for any dynamic needs of the pages.

As it is useful to allow the site to run as a dynamic site as well as the static version, I replace all WOHyperlinks with the following contruction (from the wod file):

```
Generic1: WOGenericContainer {
  elementName = "a";
  href = nodeHref;
  invokeAction = nodeLink;
}
```

These two methods have to be sensitive to context, and know when to generate a dynamic, WOHyperlink style, link, and when to generate a static link. In my administration application, used for creating and editing the site, I include a flag that is set according to if we are viewing (dynamic) or generating (static) the site.

```
public String nodeHref() {
    if (sessionViewMode()) return context().componentActionURL();
    return aNode.urlPath();
}

public CustomTemplate nodeLink() {
    if (aNode != null)
        return pageWithName(aNode.name());
}
```

The call to pageWithName() takes a component name from the node EO; again, this can be replaced with a more intelligent call that sets values in the component, etc.

The generic container build an <A> tag with an href that will either contain the appropriate static url, or will act exactly as a WOHyperlink, depending on the value returned by sessionViewMode().