AddOnMechanism 2fXmlUiAddOnIdeas

XML UI

The XML UI will be using the "sub sitemap" feature of cocoon to accomplish a plugin architecture. However this approach could be greatly increased if there was a build time process to ease the administrators' burden in configuring these files. I would like to have an XML configuration file that was just as simple as say include this plugin, not this one, etc. Then using that input a build time process could generate the cocoon specific sitemaps.

The main plugin.xmap would reference each individual plugin. In this example there are two plugins each assigned to separate urls, communityList, or the generic plugin2. This file is a bit complicated but it basically performs the current function of the web.xml for servlet inside of cocoon.

```
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
  <map:pipelines>
    <map:pipeline>
      <map:match pattern="communityList**">
        <map:mount uri-prefix="test1" check-reload="no" src="plugins/CommunityList.xmap"/>
      </map:match>
      <map:match pattern="plugin2**">
        <map:mount uri-prefix="" check-reload="no" src="plugins/plugin2.xmap"/>
      </map:match>
    </map:pipeline>
  </map:pipelines>
</map:sitemap>
```

Then an individual plugin would look something the following. The basic idea is that a plugin pulls together the various resources of the product. In this case there is one thing that is going on, the community list is put on the page and then a theme is applied.

```
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
  <map:components>
    <map:transformers>
      <map:transformer name="CommunityList" src="org.dspace.app.xmlui.CommunityList"/>
    </map:transformers>
  </map:components>
  <map:pipelines>
    <map:pipeline>
      <map:match pattern="**">
        <map:generate src="../protodocument.xml"/>
        <map:transform name="CommunityList"/>
        <map:transform name="applyThemes"/>
        <map:serialize type="xml"/>
      </map:match>
    </map:pipeline>
  </map:pipelines>
</map:sitemap>
```

Separate App build processes

Currently there is one DSpace build file that will build all applications and the core of dspace. This logic could be divided into individual pieces for easier processing. Right now I have an experiment using the XML UI with this approach but I will describe the example below as if it was the JSP UI. Each DSpace application (JSP UI, XML UI, OAI UI, METS exporter, etc...) would be separated into a top level directory. Inside that directory would be everything needed for that application including source code. For the JSP UI this would include all the java source code currently under "org.dspace.app.webui.*". Also the directory would include an ant build file specific to that application, each of these build files would then be imported by the global ant build file.

The global build file:

- Would compile the core of dspace.
- Set the classpath
- Update and install the database
- Update configuration

Then each application specific build file:

- compile the source code specific for the application
- Build the wars (if it's a war that it's building)
- Perform any build functions that are used by the application.

Since each build file is separated into their respective components targets would become a hierarchy. So the build_wars target would be jspui.build_wars, xmlui.build_wars, oai.build_wars, etc. There would be a global build_wars that would search each of the sub applications and apply the specific *. build_wars target for each applications.

This approach provides two benefits:

1. Increases the distinction between core and non core code by separating them physically on the file system.
2. Removes unused classes from specific builds, i.e. the OAI war includes all the servlets used by the JSPs.
3. Makes it easier to support multiple interfaces, just build the one that you're going to use not all of them.