Restructure Trunk Projects

Technical Refactoring and Architectural Proposals

This is a page of possible technical refactoring proposals for moving the trunk forward towards greater modularity and plug-ability.

Refactor all Applications out into modules directory

Refactoring all dspace-xxx modules out of trunk and into separate projects within the modules directory will allow us to begin asynchronously releasing updates to those modules independent of one another. Allowing existing releases to upgrade selectively to features and enhancements that they need without having to patch their own codebases to do so.

The restructuring would look like

previous path	new path under http://scm.dspace.org/svn/repo/modules/
http://scm.dspace.org/svn/repo/dspace/trunk/dspace-api	modules/dspace-api/trunk
http://scm.dspace.org/svn/repo/dspace/trunk/dspace-xmlui	modules/dspace-xmlui/trunk
http://scm.dspace.org/svn/repo/dspace/trunk/dspace-jspui	modules/dspace-jspui/trunk
http://scm.dspace.org/svn/repo/dspace/trunk/dspace-oai	modules/dspace-oai/trunk
http://scm.dspace.org/svn/repo/dspace/trunk/dspace-sword	modules/dspace-sword/trunk
http://scm.dspace.org/svn/repo/dspace/trunk/dspace-lni	modules/dspace-lni/trunk

Adopt usage of dspace-pom and eliminate dspace-parent

New Maven POM projects exist for DSpace in the modules directory, these pom eliminate a bottleneck in the dependency mechanism defined in the dspace-parent pom in favor of allowing individual projects manage their dependency versions. It is released separately from the DSpace release process so that it can be utilized before the official release to base new modules on.

http://scm.dspace.org/svn/repo/modules/dspace-pom/tags/

Doing this increases the flexibility to define and override dependency versions used when customizing DSpace instances in the wild. And depending on this pom provides all the basic details for the build and release process of maven without forcing specific dependencies onto the components that use it.

Promote using the non-source release

If users work with the source release as the default, it defeats the whole process of providing a build and development process that is modular. end users still customize the classes directly and still attempt to alter core classes that they should not be touching.

Working with a binary release clarifies which calles should be allowed to be overridden in DSpace and promotes using configuration and pluggin interfaces in DSpace over modifying and recompiling code in place. Moving the dspace-xxx modules out of trunk further enhances this dichotomy. And allows for a clear separation of User customization and DSpace development direction.

Refactor DSpace API into separate modules where appropriate

Why, because we cannot maintain modules independent of core that depend on a specific core release version and then release them as part of core. It may sound convoluted, but this is an example of what has happened with dspace-stats in DSpace 1.6.0. Our initial goal was to maintain the stats packages independently, depending on a specific release of DSpace services. Individual user interfaces would have still resided in the dspace-xmlui and dspace-jspui cores. But the goal was to exemplify module development occurring outside the core and easily being added into it forthe release process.

Why did this fail? This failed due to circular dependencies between dpace-statistics and dspace-api. This was something that the work in DSpace 2.0 sought to eliminate by allowing the creation of API separate from implementation that were not dependent on a central core with all the features in it. An example of this being successful is dspace-discovery. Where once it is finished, it should be only dependent on dspace-services and solr, eventually int he future it will not even be dependent on dspace-api.

DSpace API is a entangled mass of conflated functionality. It is highly recommended that DSpace API be separated apart in terms of functionality to support greater separation between content model, applications and utilities in. Proposed areas of functionality need further analysis but look like the following:

- Applications: DSpace Applications are generally found under the package org.dspace.app and generally other package should not be dependent on these. It is recommended that they be broken off where appropriate and repackaged individually (statistics, import, export, mediafilter, bulkexport, etc) The goal in separating these out is to promote development fo new applications without dependency on the dspace core release cycle and reduce the need for core releases to add application enhancements to DSpace.

- Core: DSpace Core is made up of core, content, event and a number of other packages, these packages should be reviewed with a dependency analysis and repackaging in to separate maven projects with separate release cycles will improve our ability to provide updates to these packages outside a "core" release cycle"

The ultimate goal in separating out this functionality it to get the core of DSpace clearly defined as a set of core Service API backed by an encapsulated implementation. DSpace Applications should only be dependent on the Services API defined and not the actual classes implementing the backend functionality.

Outlined below are some of the possible directions for refactoring once we begin this process of refactoring.

Refactor DSpace ConfigurationManager to use Configuration Service.

This proposal is centerd around coninuing an effort to break apart the exisitng core DSpace modules to support further modularity. Firstly I will present the problem.

Refactor DSpace EventManager to use EventService

DSpace EventService is currently just processing usage events, it can also process all other events, EventManager should be dropped in favor of EventService , all usages of EventManager should be replaced with EventService and event Consumers rewritten to be EventListeners attached to the EventService.