EvolutionToServiceLocatorProposal

N.B. The benefits listed here could also be achieved with an IoC pattern – JimDowning

I started thinking about refactoring the current API when I tried to add some unit tests. Because all the methods are essentially libraries loaded by the classloader (on specific classes), it's very difficult to extract a 'unit' to test it, without the unit being the whole app. Ideally each of these libraries should have their interfaces extracted, and implementations should be obtained by a central factory. I realised also that the ubiquitous Context object provides us with a handy place to store the reference to the service locator.

Proposed Solution

The service locator is described by an interface giving access to all the service interfaces. It's implementation is created from a configuration parameter (for example the class name). This allows you to replace any DSpace service implementation you wish by implementing your own ServiceLocator. The DefaultServiceLocator should not be final, to allow other implementations to use most of the default service implementations, but change one or two easily. The configuration of the service locator implementation could be separate from dspace.cfg, or it could all bootstrap up from dspace.cfg (the rest of configuration management will probably be a service)

The application environment will probably wish to cache the service locator implementation as a singleton in some way or the other. The command line tools could use a static property, the web application would be better off caching in a ServletContext attribute.

An extra step in each application flow is to set the call Context up with a (possibly cached) service locator. All applications and services can then obtain other services through the service locator on the context.

With only a little clumsiness with the naming it should be possible to leave the static methods where they are (although deprecating them may be wise), and change their implementations to delegate to the new service implementations via the service locator.

Benefits

- Unit testing will be a more viable proposition.
- The abstraction of the service locator interface allows for a very basic method of prototyping modified functionality into dspace.
- Potentially backward compatible with current API

Potential Problems

- All current static methods will need to take Context as a parameter. If they currently don't, the API will have to be modified.