DSpace AddOn and Component Management System
what and why

• What is an addon or component?
  – a third-party feature
  – a localisation
  – an official DSpace component

• why do we need them?
  – ease creation of new tools
  – improve modularity and plugability of DSpace
  – allow for multiple alternatives for repositories
  – ease management of customised instances
Aspects of the AddOn Mech

- New build/install/update process
- Component management system
- Component versioning
- Skeleton component for quick build of new tools

Note: this is a prototype, and feedback is welcome and encouraged
Prototype build process
Component Management

- Component Versioning
  - useful for managing schema and code installation
  - useful for managing updating and rollback
- Installation order
  - some aspects of the code overwrite the installed components before them
  - important for schema, UI components, and configuration
- Component Registry
  - components.xml holds info on registered components, their version, and installation order
  - database registers which components (and versions) have installed schemas
Challenges: configuration

- XML file merging solution adapted from another Open Source project (Grouper)
  - successful with dspace-web.xml
  - TODO for dspace-tags.tld, input-forms.xml and any other XML configuration files
- Properties files
  - use of Java Properties class
  - working for dspace.cfg and Messages.properties
  - TODO for non-unique key property files (e.g. dstat.cfg) and multiple Messages_X.properties
Challenges: to patch or not to patch

- Each component has own JAR file
- Encourage the use of good modular programming rather than overwriting other code
- Encourage the use of configuration based alternatives (e.g. using the PluginManager)
- Allow DSpace to move slowly in the right direction rather than using a quick fix which may be damaging in the long term
- Allow compilation of subsequent components against installed ones by placing the JAR in the live library immediately
Challenges: JSPs

• Original method: JSPs in jsp/local are copied over their counterparts in build/jsp during build

• Doing this with components is highly sensitive to installation order:
  – components can override default DSpace JSPs
  – components can override each other's JSPs
  – institutions will have their own localisations
  – Therefore, risk of conflicts (no solution yet)

• Component Manager controls installation order
  – DSpace first, components next, localisations last
  – This is not enforced, only recommended
Challenges: other UI issues

• Common UI components
  – navigation: lots of tools will add navigation items, so cannot rely on JSP overwriting
  – style sheets: tools should add their own styles, not override the default.
• Possible solutions:
  – XML driven navigation in prototype. Will this make things easier? How do we add new items at install?
  – Similar solution for stylesheets?
• Future: What impact will Manakin have on this process?
Challenges: database management

- Updating from an arbitrary point to the latest
- Single schema files vs multiple schema files
- Registering schema versions with the Component Manager
- Registry loaders (e.g. Bitstream Formats, Metadata Registry, and now Metadata Schema Registry)
- Schema files should not load any actual data
- Database destroy scripts required
  - Uninstallation process is not yet clear
Default Installation Walkthrough

```
built.xml

ant -Ddir=/dspace init
```

Diagram:
- Source
- copy
- `/dspace`
  - `some live dirs`
  - `components`
Default Installation Walkthrough

build.xml
ant build
components/dspace/build.xml
ant install
jsp
src
config
build
deploy live dirs
Default Installation Walkthrough

1. `ant install`
2. `load schema`
3. `load registries`
4. `prepare configs`
5. `main build directory`
6. `build webapps`
7. `components/dspace/build.xml`
Component Installation Walkthrough

build.xml

ant - Dsrc=/addon ingest

copy

Source

copy

component

addon

identify

component.properties

name

addon

rollback/addon

addon
Component Installation Walkthrough

build.xml

ant - Daddon=addon build
components/addon/build.xml

ant install

jsp
src
config

build

Therefore, the default DSpace build process assumes addon=dspace when invoking ant
Component Installation Walkthrough

```
build.xml

ant - Daddon=addon register
```

Diagram:

- **ComponentManager**
  - `insert` to **components.xml**
  - `dspace`
    - version = x.y.z
    - install order = 1
  - `addon`
    - version = a.b.c
    - install order = 2
Component Installation Walkthrough

```
build.xml

ant install

ComponentManager

foreach [component]

components/[component]

prepare configs

build webapps

load registries, etc.

load schema

database

components/[component]/build.xml

ant finally

main build dir
```
Component Skeleton

- build/
- config/
  - language-packs/  
  - registries/  
  - dspace.cfg
- etc/
  - schema.sql
  - schema-destroy.sql
  - dspace-web.xml
- jsp/
- lib/
- src/
- build.xml
- component.properties

created when necessary

Messages.properties

e.g. dublin-core-types.xml

component config options

db schema create

db schema destroy

servlet mappings, etc...

for additional JARs etc...

component source tree

component build file (templated)

component properties file
Component Skeleton build file

- Primary Targets (called by the master build file):
  - install – compile the sources into the addon build directory *(implemented)*
  - finally – finish up the installation. For example, load the database registries. *(unimplemented)*

- Secondary Targets (called by the primary targets)
  - build_structure – create the build directory sub-structure *(implemented)*
  - compile – compile the source code into the build directory *(implemented)*
  - make_jar – make the source code into a JAR file *(implemented)*
  - configs – move the configs into the right places *(basic case implemented)*
  - jsps – put the jsps in the build directory *(implemented)*
Component properties

# Required properties:

# The name of the component. This will be the directory in which it is stored
# in the DSpace source tree, so please ensure that it is compliant with
# directory name standards. Ideally only use alphanumerics (avoid
# spaces). This does not have to be the name of the component as it is commonly
# known, it is for system use only.

component.name = addon

# the version of the component. Components are required to use at most a
# three level version numbering scheme. The versions should be expressed
# as numbers only, in the usual order. This is to allow comparisons of which
# is most recent in the version chain by the DSpace system

component.version.major = 1
component.version.minor = 0
component.version.subminor = 0
towards a first Stable Release

- More XML merging (tags and possibly input forms)
- Reference Implementations
- Testing
  - making and installing addons
  - upgrading from the current DSpace
- Tidying up
  - better Exception handling
- Documentation
  - inline installation help documentation
  - developer documentation
beyond a Stable Release

• Translated message file merging
• Component dependency warnings/resolution
• Component version rollback
• Multiple webapps
• Modular documentation management
• Anything else that is suggested and useful ...
• Your contribution ... please get involved
References

- Manakin XML UI project http://wiki.dspace.org/Manakin
- Grouper (XML merging code) http://middleware.internet2.edu/dir/groups/grouper/
Thanks for listening

Richard Jones

Universitetsbiblioteket i Bergen
Tilvekstavdeling
Nygårdsgaten 5
5015 Bergen
Norway

richard.jones@ub.uib.no
http://bora.uib.no/