OASIS Development Stream

Create the OASIS development stream in trunk

A. Proposal for the Stages

Stage 1: Foundation under Construction

Community expectations

- Focus on cleaning up the core areas (such as assembly model, contribution, core, core-spi, and implementation-java).
- Good amount of refactoring may occur
- Not ready for adding new functional features before the core is settled. Don't rush to bring it up to the same functionalities as 1.x.
- Code will be unstable and/or broken for a couple of weeks

1. Start from empty trunk (after the current trunk is branched off to 1.x)

- This gives us the opportunity to start from fresh without being contaminated by what we have in 1.x.

2. Establish processes in the project to keep the code of the foundation clean, simple and interesting to work with.

- Some guidelines for package visibility, cross module dependencies, SPI principles
- We are devloping a guide to help develop Tuscany modules with OSGi (Please help enhance it as you try out)

- http://cwiki.apache.org/confluence/display/TUSCANY/OSGi+Aware+Programming+in+Tuscany

3. Get the right set of tools from sca-equinox branch to help us enforce and maintain modularity with clean SPIs.

- We already have a fairly good developer story in sca-equinox branch:

- Leverage Eclipse PDE to develop Tuscany modules as OSGi bundles
- Adopt Eclipse JDT compiler with OSGi bundle resolution in maven build

4. Copy modules from sca-equinox branch into trunk

- Most of the modules in sca-equinox branch has been converted into OSGi bundles
- We have fixed the access violations reported by PDE OSGi validation
- Some level of clean-ups have been done
- No functional deviations from the modules in the current trunk (1.x)

5. Perform thorough cleanup for the core set of modules

- Identify a subset of the modules (i.e., the core) as the focus for the cleanup as proposed below in section C. (Note: Having other modules in the trunk so that they can be refactored with the core to minimize porting efforts).
- Some of the items can be progressed independently. We may have to accept the fact that some require good coordination or even controlled mode.

Stage 2: Functional Bring-up

Community expectations

- Stable SPIs
- Stable core to integrate extensions
- Good foundation to add new functions

1. Merge changes from the 1.x branch into trunk
2. Further apply the OSGi modularity to extension modules
3. Bring up the unit tests
4. Bring up the samples, itests, vtests, demos and tutorials
5. A redesign of some problematic or over-complicated areas like policy for example (that's a good one to redesign as it's also impacted by the OASIS changes).

B. Some TODO items for Stage 1

1. Use the Import-Package/Export-Package OSGi headers for Tuscany modules to define cross-module dependencies and enforce SPI contracts

- Developers will be responsible for authoring and maintaining META-INF/MANIFEST.MF files

2. Reduce the Import-Package/Export-Package to be only a minimum set of SPIs

- Some implementation classes are still referenced across modules, for example, org.apache.tuscany.sca.databinding.impl package is exported.

3. SPIs have not changed for over 18 months and a new base gives us a chance to clean up the SPIS that have been polluted with workarounds and non-optimal changes.

4. Extract SPI layers for some modules

- For example, tuscany-core has quite a few packages that are exported. We need to define a SPI layer if these exports are required.
- implementation-java: extract out the annotation processing so that they can be reused for introspecting other java-based implementation technologies such as Spring or EJB
- implementation-java-runtime: extract out the IoC
- add common modules such as common-java and common-xml to host java or XML related utilities

5. Cleaner code, a pass through all the modules to add comments, adjust the visibility of classes/methods, sort between static and instance methods, remove deprecated/unused code, organize imports, etc.

- As we went through OSGI enablement, we discovered quite a few cases where unused imports brought in unnecessary dependencies. It would be good to adjust visibility of classes/methods, clean up static and instance methods, add comments for maintainability.

6.The project has accumulated a lot of dead code over time. The dead code makes it difficult to find your way through the code base and causes more complication.

- For example, we have different ways to read/resolve SCA contributions
- Rationalize and clean up the SCA contribution classloading strategy

7. A consistent separation of extension modules into model and runtime

- For example, implementation-spring should be split into implementation-spring and implementation-spring-runtime

8. Convert test cases from Junit 3 style to Junit 4 style

- Avoid redundant setup/destroy steps to reduce build time
- Cleaner way to ignore tests