Redesign for more standards including Maven

Redesigning for more standards including Maven

NOTE: This is a work-in progress as of 1/28/2010

This is a proposed redesign of the layout in SVN and the way we use Maven, to enable broader use of standard tooling and approaches for development and release.

As the new design takes shape over time, it will be documented here:
Maven use design as top level project

This redesign could be part of our move to a TLP (top level project), assuming we graduate 🙂.

Goals

Fix things that work poorly now.

- RAT reports - currently run on entire assembly, which means we have to unzip it, and run on that.
  - should run on pre-zipped things
  - Therefore, try to run on individual projects, not on distr

- Non Hierarchical POM layout (flat layout) causing problems
  - Release plugin doesn't work
    - may be fixed in latest version of Release Plugin
  - Assembly plugin "modules" doesn't work
    - because it requires the aggregator pom to be in the parent directory of the modules, I think
  - m2eclipse works nicely with hierarchical layout
    - prototyped this already, and can make things work with hierarchical directory layout
  - Simple things like checking out all of uima - uimaj, uima-as, sandbox, (uimacpp - work on this later) from SVN with one export, and then being able to build, doesn't work
    - Current extract & build requires copying over / merging separate SVN checkouts

- Source release doesn't match SVN
  - should be just a zipped up version of the svn, or generated using the source mvn plugin.
  - source assembly should be (if possible) just the sources zipped up
    - may need to have a SVN structure that separates things not in releases (e.g. uimaj-internal-tools)

- Building from checkout - workspace and SVN layout don't match, so after checkout (via command line from a top level node like sandbox) you need to copy this into the same dir as another checkout (e.g., uimaj).
  - This was caused by earlier Eclipse needing to have everything in one level.
    - Now you can have the non-java projects imported into Eclipse, containing other java projects as subdirectories, and simultaneously, also import those Java projects into Eclipse as other projects
      - M2Eclipse does this, for example

- Many parts don't change from release to release - avoid re-releasing these
  - make more use of component part versions
  - support add-on projects being individually released

- POM hierarchy mixes up aggregation with inheritance
  - Follow new "best practice" of having at least some of the super POMs have integer version numbers, and not changing these very often (see TBD web ref)

- POMs used for overlapping purposes
  - Consider a superpom hierarchy that separates these separately, for easier maintenance
    - maven tooling dependency version
    - component dependency version (if appropriate)

- Make project versioning work with Release plugin
  - Substitute a more standard way for our use of "properties" for version numbers.
    - This should allow our POMs to have less version dependence.

- Balance maintenance-motivated DRY (don't repeat yourself) with obfuscation
  - sometimes having things factored so as to inherit from things and/or use properties set elsewhere in complex ways makes future readability / maintenance difficult.
  - using **properties** for version information in some cases is triggering warning messages, in m2eclipse, and in the next version of Maven (maven 3) - saying this is not good practice, and may not be supported in the future.

- Figure out Maven's approach to properties
  - It seems that there is a mechanism for "releases" that captures the settings of properties used in build the release, for reproducibility
  - It also seems there is a concept in the Release plugin where it substitutes actual values of properties when it makes the "tag", and then puts back the ${xxx} variable form when it updates the trunk. This allows the tagged item to be more "independent" of other components - for instance, when it is located in a Maven repository as a downloadable artifact.

Support future incorporation of Continuous Integration (CI)

Add Website support automation

Currently the website is maintained completely manually. More standards could enable automatic creation of various developer reports, updating the download page as part of the release process, etc.

Design

After thinking that redesigning svn layout was important, I now think that's not the case, but what is important is to change how the build works, as follows:

- make each component independently buildable
    - This is the main "fix". There are several things in the current build that violate this.
        - We have a custom docbook build - that uses a "checked-out" docbook tool project, that currently has to be in a certain directory hierarchy relationship with the project being built. This is fixable by no longer using our own docbook tooling, and instead using the recently developed docbkx maven plugin.
        - We also have dependencies on parent POMs. This can be fixed by making those dependencies use poms in the maven repository (or the local repository). This requires that any updates to these POMs be put in either the local repo or separately "released".
            - I've noticed that many projects have moved to this style, and name their parent poms with release numbers like "1", "2", etc. For instance, the common Apache "parent" pom is at release "7".
            - It would be good to reduce POM changes in parent poms by isolating the POMs to particular functions. For instance, we currently often mix up the "aggregator" pom function (<modules>) with being a parent. The aggregator will change with every release - it will include particular items at particular levels for the release, while the parent poms will likely change less often.
        - We also have dependencies on particular files, such as common license / notice files. These kinds of common resources are now being handled by the maven remote-dependency plugin. The Apache Parent POM does this, for instance, and we can move to using this same style of dependency, to remove any requirement for particular checkout hierarchy.

Because of this, I don't think that any change is **required** in the current svn tree; we can use other use cases to motivate incremental changes to the svn tree structure, as needed.

- One possible change is to create a new top level svn node holding non-released tooling (e.g., the projects uimaj-internal-tools and uimaj-jet-expander) - and move these there. This will enable source releases to be just the zipped up sources of the svn checkout (after maybe moving some license & notice files around as needed).
    - a new top level svn node called "build" has been created, and holds build-related things such as maven parent poms, resources needed by the build processes, etc.