best practices - version management in multi project bu

Issues to resolve

- We want to provide a single place for dependency versions to be defined for a multi project build.
- We need to be able to separate metadata version from application version in a parent POM, yet still define them both in the parent POM. The application version should be referenceable for use in the child POM's version element.

Dependency Versions

Option 1

Using the standard dependencyManagement element to specify the versions.

Possible limitations:

- Might be that you want to use the version string somewhere else for whatever reason. You could reference the dependency but that would probably be pretty verbose.

```
<project>
  ...
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.foo</groupId>
        <artifactId>foo</artifactId>
        <version>1.0</version>
      </dependency>
     <dependency>
        <groupId>org.bar</groupId>
        <artifactId>bar</artifactId>
        <version>2.0</version>
      </dependency>
    </dependencies>
  </dependencyManagement>
  ...
</project>
```

Option 2

This option has each version used specified in a property and that property is used in the dependencyManagement element and can be used anywhere else.

Possible limitations:

- Any tooling made to help manage dependencies would be dealing with free form properties

```
<project>
  ...
  <properties>
    <fooVersion>1.0</fooVersion>
    <barVersion>2.0</barVersion>
  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.foo</groupId>
        <artifactId>foo</artifactId>
        <version>${fooVersion}</version>
      </dependency>
     <dependency>
        <groupId>org.bar</groupId>
        <artifactId>bar</artifactId>
        <version>${barVersion}</version>
      </dependency>
    </dependencies>
  </dependencyManagement>
  ...
</project>
```

Option 3

Using the standard dependencyManagement element to specify the versions and generate properties based on the dependency declaration

Possible limitations:

- might appear as magic that properties are just appearing, but could make a project-info goal to print out available properties

```
<project>
  ...
  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.foo</groupId>
        <artifactId>foo</artifactId>
        <version>1.0</version>
      </dependency>
     <dependency>
        <groupId>org.bar</groupId>
        <artifactId>bar</artifactId>
        <version>2.0</version>
      </dependency>
    </dependencies>
  </dependencyManagement>
  ...
</project>
```

jdcasey

The most EL-neutral syntax for referencing such a built-in property might be something like:

```
${dependencies["org.foo:foo"].version}
```

One note on using dependencyManagement information for POM interpolation: depMgmt info is only available when directly referenced by the POM, and the injection of depMgmt information currently takes place after interpolation IIRC. It's just another detail we'll have to deal with in order to enable this.

See Expression Access to POM List Elements for design discussion.

JIRA reference

Application Version

It would be good to allow parent POMs to change their version when metadata changes, yet still keep some other application-level version constant, so that child POMs can reference it. This way, the parent POM can have a version that is a serial number, denoting the number of changes since it was created...this might be roughly equivalent to its SCM revision number, I suspect, but that's tangential. At the same time the parent version may change to accommodate new developers, alternate SCM URLs, new site info, etc., we want to be sure the version used to define the current version of the application **does not** change. That way, child projects don't change their version every time metadata nonessential to the build changes in the parent.

Creating the ${applicationVersion} concept

So, the parent might look like this in today's world (where we only have POM-properties to play with this concept):

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.someproject</groupId>
  <artifactId>project-root</artifactId>
  <version>1</version>
  <name>Some Project (Root)</name>

  <properties>
    <applicationVersion>1.0-SNAPSHOT</applicationVersion>
  </properties>

  ...
</project>
```

and the child might look like this:

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.someproject</groupId>
    <artifactId>project-root</artifactId>
    <version>1</version> <!-- ugh, this will have to update for each metadata change!
-->
  </parent>

  <artifactId>project-core</artifactId>
  <version>${applicationVersion}</version>
  <name>Some Project Core</name>

  <dependencies>
    <!-- this is a dependency within the application, so we're going to use the
         application version again. -->
    <dependency>
      <groupId>${groupId}</groupId>
      <artifactId>project-api</artifactId>
      <version>${applicationVersion}</version>
    </dependency>
  </dependencies>
  ...
</project>
```

Next steps

From here, we can get smarter about using the applicationVersion, if it's specified. For instance, we can say that as a third layer of resolving dependency versions, we're adding the ${applicationVersion}, **iff** the groupId is the same. So, dependency versions would be resolved using the following prioritization:

1. local dependency specification
2. dependencyManagement section (inherited)
3. applicationVersion, assuming the dependency's groupId is the same as the current POM's

Also, we could implicitly resolve the current POM's version using ${applicationVersion}, provided it has the same groupId as the parent POM, and the parent specifies ${applicationVersion}.

A couple of warnings

We need to make sure that only the root POM for a multimodule build can specify applicationVersion, which implies that there can be only one applicationVersion for a groupId. This is consistent with the concept of the applicationVersion, where the application itself has a strong version number, and the child projects adhere to this master version. If a child POM respecifies the applicationVersion, this is really another application (or so it would seem), and should also have a different groupId.

Along with this, we should be sure to remove the applicationVersion property from inheritance when a child POM doesn't share the parent POM's groupId. Not sure why you'd have this situation come up in an application build, but changing the groupId would seem to define that project as technically outside the application, so it probably shouldn't be able to share the applicationVersion property.