GlassFish v3 Build System

Kohsuke Kawaguchi

Goals

- Do more at development time
  - Less boiler plate code, less build script
- Facilitate automation
  - With the eventual goal of fully automating all integrations
- Work nicely with mercurial's hierarchical workspaces
  - I couldn't make this goal work. More later
- Must be usable for developing 3rd party modules
  - This includes other Sun products built on top of GF
  - This also helps component teams like Metro
- Deliver GF in more ways

Key ingredients

- Maven2
  - Enabler for "do more with less build script"
  - Enabler for automation, 3rd party use
  - Risks
    - Poor implementation
    - Horrible error messages
    - But in practice, only practical choice
- Hudson
  - Enabler for automated integration
- GFv3 maven repository
  - Build artifacts are deposited and downloaded from here
  - Enabler for network install / update center

Term Check

- Module
  - Unit of build and deployment
    - A distribution of Glassfish = a collection of modules
  - Corresponds to one maven module
  - Belongs to one SCM repository
- SCM repository
  - Place for keeping source code for 1 or more modules
- Project: a java.net project
  - Unit of presentation
  - A project might use 2 SCM repositories
  - 2 projects might host the code on 1 SCM repository

SCM Repositories

- We don't really care where repositories are created or how many
  - … except almost certainly # of repo > 1
- We don't really care if it's CVS or Mercurial
- Factors for deciding what should be a repository
  - Bigger repository is costly
  - Two unrelated teams would prefer two repositories
  - Closely related modules would prefer one repository

Module

- A GF module is a maven module
  - Roughly speaking it contains the following stuff

```
somemodule
+- pom.xml (build script)
+- src
   +- main
   |  +- java (source code)
   |  +- resources
   +- test (unit test code)
```

- Lots of APT processing
  - Metadata generation and code generation
  - All hidden behind Maven

Module

- Build will eventually produce a jar
  - Contains all the necessary metadata for runtime
  - More metadata for later builds, such as distribution build
- POM eventually extends from GFv3 super POM
  - Directly or indirectly
  - Upload super POM to central maven repo so that modules can be built without special ~/.m2/settings.xml

Module

- We don’t really care how many modules we have
  - But it will be a very large number
  - It will be open-ended as we encourage 3rd party modules
- Factors for deciding what should be a module
  - Working on too many small modules are tedious
  - Module is abstraction/grouping tool like package
  - Module is the smallest unit of deployment

Distribution Module

- Distribution module is a maven module that builds the runnable Glassfish image
  - Basically just list up modules to be included

```
<project>
  <artifactId>pe</artfactId>
  <dependencies>
    <dependency>jaxb:2.2-build-1500</dependency>
    <dependency>jax-ws:2.2-build-923</dependency>
    <dependency>ejb:10.0-build-329</dependency>
```

Distribution Module

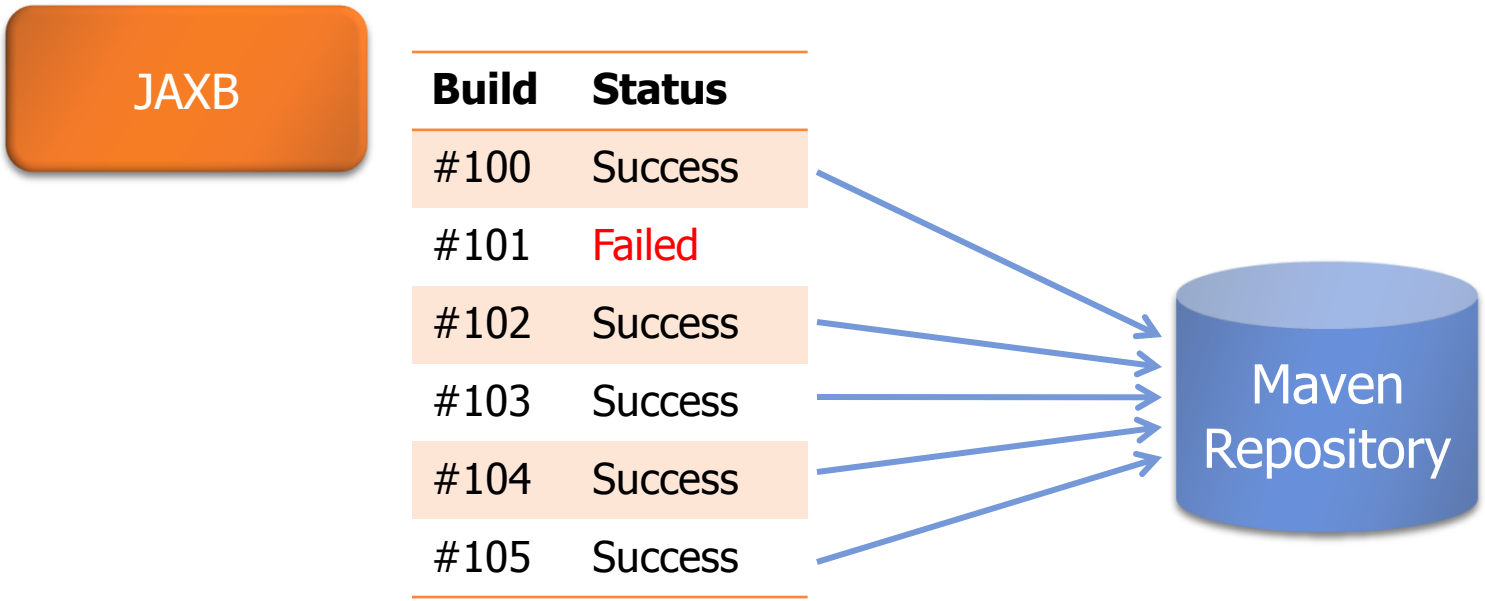
- Support inheritance
  - Not via POM inheritance but through transitive dependency traversal
  - Will be used to create bleeding-edge GF images

```
<project>
  <artifactId>pe</artfactId>
  <dependencies>
    <dependency>jaxb:2.2-build-1500</dependency>
    <dependency>jax-ws:2.2-build-923</dependency>
    <dependency>ejb:10.0-build-329</dependency>

<project>
  <artifactId>pe-webservice-bleeding-edge</artifactId>
  <dependencies>
    <dependency>pe:10.0-SNAPSHOT</dependency>
    <dependency>jaxb:2.2-SNAPSHOT</dependency>
    <dependency>jax-ws:2.2-SNAPSHOT</dependency>
```

Continuous Integration

- Continuously build modules
  - 1st line of defense against bad code
  - Builds get published to GFv3 maven repository
    - By build numbers*
    - "Garbage collection" needed to keep disk usage under control



* We should be able to do this without touching POM. Needs more research

Continuous Integration

- Run tests continuously
  - 2nd line of defense against bad code
  - "tests" maybe unit/SQE/TCK tests, or maybe integration build of another module with this new bit

JAXB

Build	Status	Tests
#100	Success	All pass
#101	Failed	
#102	Success	5 failed
#103	Success	All pass
#104	Success	All pass
#105	Success	(in progress)

Continuous Integration

- When builds pass certain bars, Hudson updates other POMs to pick up new build
  - What POMs to get updated will be configurable
  - That bar might be "never", meaning manual integration

Build	Status	Tests	Action
#100	Success	All pass	Picked by JAX-WS
#101	Failed		
#102	Success	5 failed	
#103	Success	All pass	Picked by JAX-WS
#104	Success	All pass	Picked by JAX-WS
#105	Success	(in progress)	

Continuous Integration

- Propagation will work as further bars
  - e.g., update to JAX-WS POM will cause new JAX-WS builds and its test runs, whose results will feed back to JAXB

Build	Status	Tests	WS tests
#100	Success	All pass	All pass
#101	Failed		
#102	Success	5 failed	
#103	Success	All pass	3 failed
#104	Success	All pass	All pass
#105	Success	(in progress)	

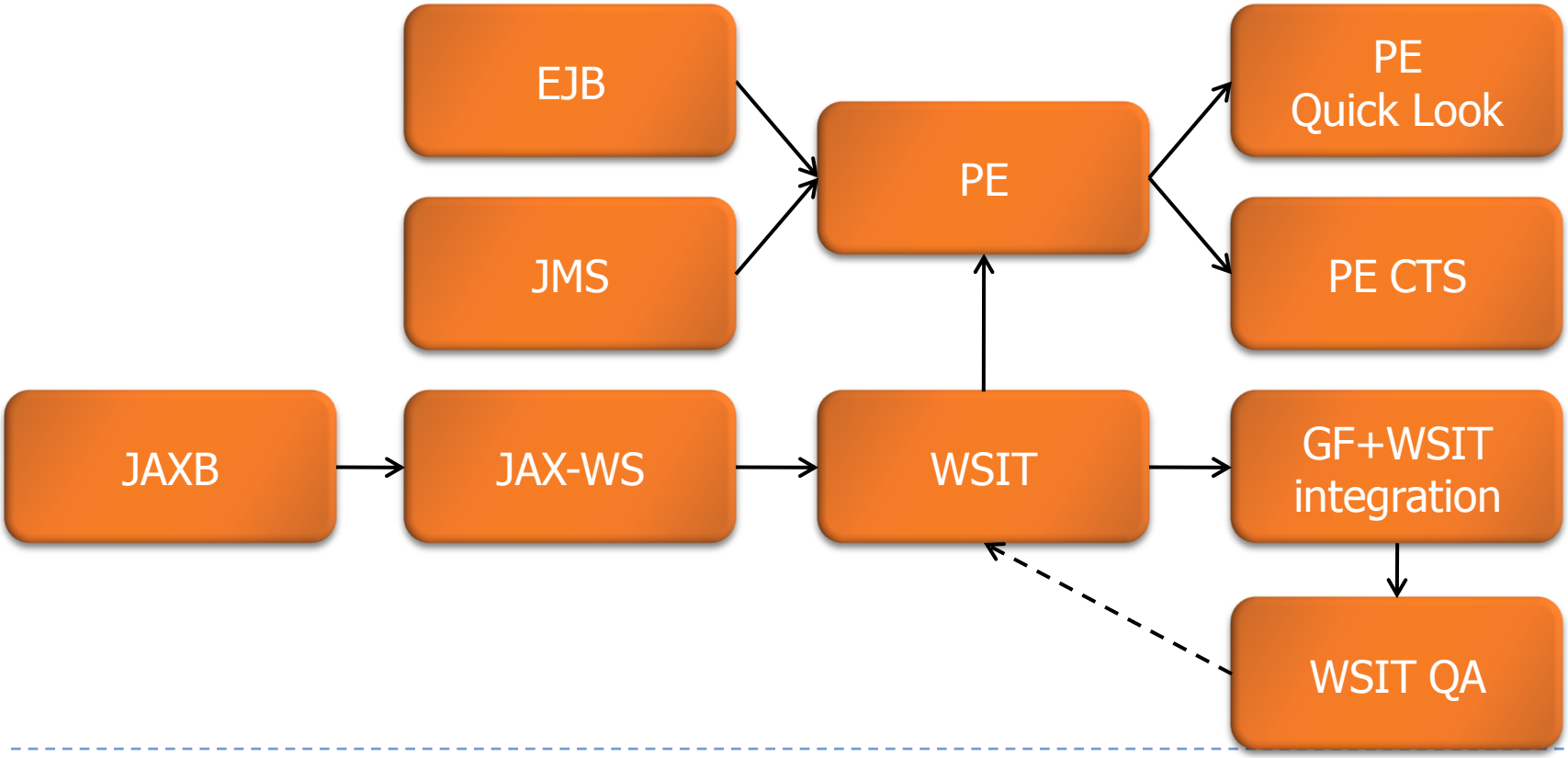
Continuous Integration

- ... and those feed backs can be used to trigger further propagation

Build	Status	Tests	WS tests	GF
#100	Success	All pass	All pass	Picked by GF
#101	Failed			
#102	Success	5 failed		
#103	Success	All pass	3 failed	
#104	Success	All pass	All pass	Picked by GF
#105	Success	(in progress)		

Recap: Mental Picture of Continuous Integration

- Think of this as a graph of projects where builds (hence changes) propagate through controlled fashion



Tests

- Speed of change propagation depends squarely on tests
- This proposal does not force any changes, but fully automated, fast-running tests will make a real difference
  - Not just speeds in which changes propagate, but more importantly keeping qualities high constantly and allowing developers to take a larger risk
- The same goes to component specific tests
- There's a lot of rooms for taking advantages of this in tests
  - Needs further discussion

Module Developer Experience

- Build a module

```
$ hg clone http://hg.glassfish.java.net/ejb/
$ cd ejb
$ mvn install
```

  - dependencies downloaded from maven repository
- Debug

```
$ MVN_OPTS=-Xrunjdwp:… mvn gf:run
```

  - This will launch Glassfish inside Maven with …
    - modules listed in some build of some distribution
      - configured in this module over by CLI argument
    - plus the current module

Module Developer Experience

- You just need to check out modules you are working
- Maven modules can be opened by any IDE
- Making changes across modules
  - Modules on different SCM repositories need to be checked out individually
  - May have to invoke maven multiple times to build all relevant modules
    - that is, if they don't have the common parent POM
    - in such case, POMs needed to be updated manually to use SNAPSHOT dependency
  - This is not as easy as it should be

3rd Party Module Developer Experience

- I should be able to write a few modules
  - Write one maven module per one GF module
  - Build them by "mvn package"
  - Run them with "mvn gf:run"
  - Install the resulting jars on any GFv3 installation
    - GFv3 needs to provide a directory to drop them
    - Update center to further automate distribution and installation
    - Very much like how you handle NetBeans modules

3rd Party Module Developer Experience

- I should be able to create custom distribution
  - Write a custom distribution module
    - Derive from existing distribution and add more modules
  - Distribute resulting image
  - The same automation scheme would work for this
  - Bring what MyEclipse does to Eclipse to GFv3
- We need a lot of custom distributions internally, too
  - e.g., WSIT SQE needs to have a GF image with latest WSIT to run tests

Release Engineering

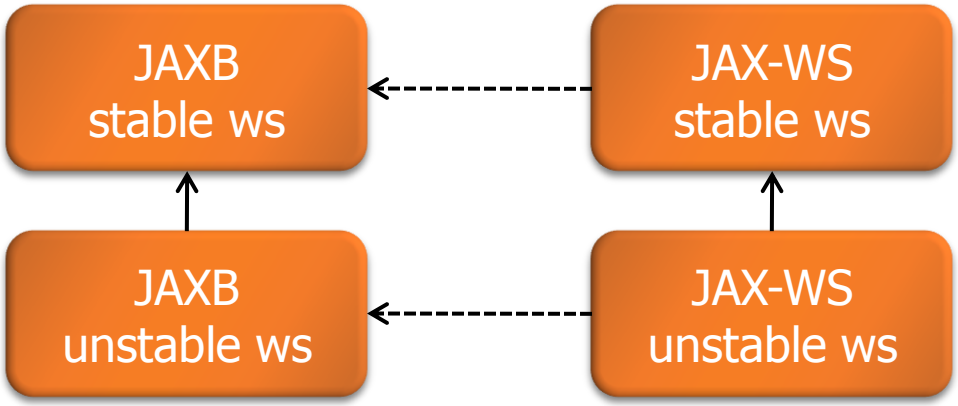
- No separate RE outside continuous integration
  - Qualified CI builds will replace promoted builds
- No single command will build the entire GFv3 from scratch
  - Why? Think about …
    - bunch of components picked up from maven repo as binaries
    - multiple repositories spread all over the places
  - GFv3 will be more like a federation of loosely coupled modules
  - We are still building everything from the source
    - It's just that we are not doing this all at once

Issue: This Proposal and Mercurial

- Not taking full advantages of workspace hierarchy
  - IOW, this proposal just works fine with SVN/CVS
- Why?
  - Dealing with multiple lines of builds from single module difficult in Maven
    - In particular, declaring dependency on them is difficult

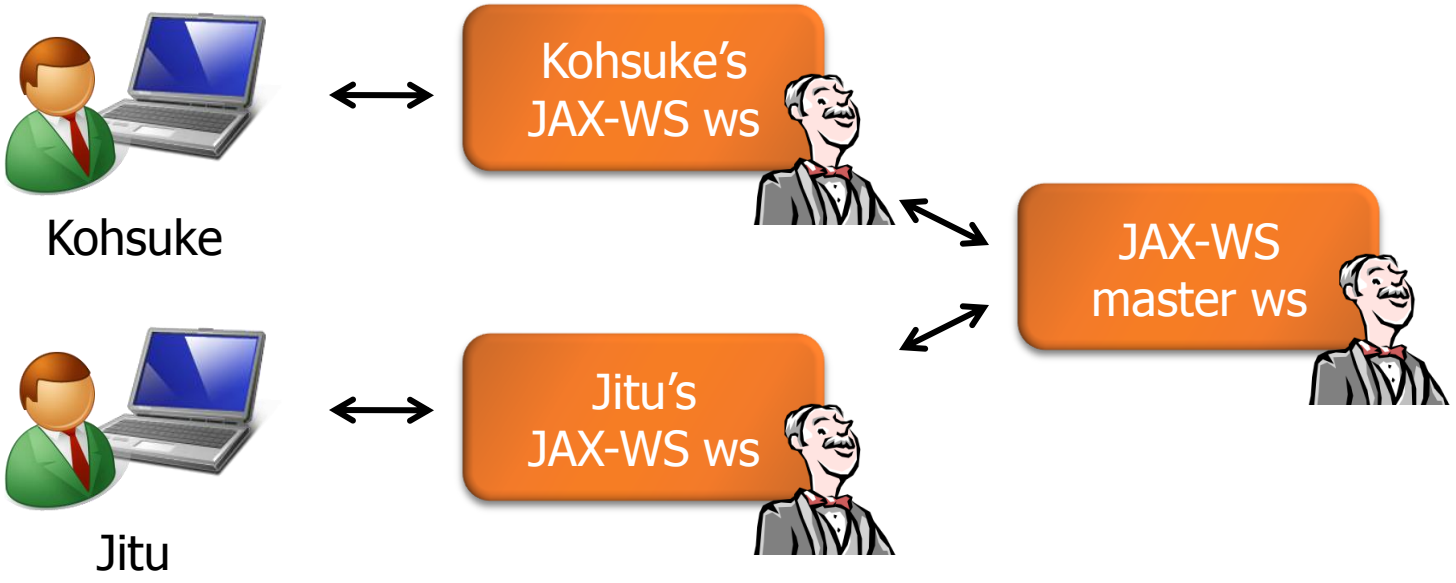
Issue: This Proposal and Mercurial

- Why is that hard? Let's assume…
  - stable JAX-WS depends on stable JAXB, unstable JAX-WS depends on unstable JAXB
  - Now all tests passed in "JAX-WS unstable" and so you pushed your changes upstream
    - Which JAXB stable build are you going to depend on?



Issue: This Proposal and Mercurial

- It doesn't mean we can't use workspace hierarchy
  - Modules can locally create them and use them
  - In fact could be a great "personal build" mechanism to avoid running any tests on your local machine whatsoever



Issue: This Proposal and Mercurial

- This proposal still achieves the same goal
  - Downstream projects can avoid picking up unstable bits
  - It just does so in a different way
- After all, when you integrate lower workspace to higher workspace, how do you know the code is good?
  - It's better to let tests run, than to rely on humans

Issue: This Proposal and Mercurial

- So I stopped worrying
  - Whereas workspace hierarchy spreads builds of different quality over spatial dimension, this proposal spreads them over time dimension
  - Keeping track of where the changes went is easier with single line
    - I just need to say "I fixed the bug you reported in jaxb #123"
  - This proposal is closer to how we do things now

GlassFish v3 Build System

Kohsuke Kawaguchi