GlassFish v3 configuration mechanism

Kohsuke Kawaguchi

Problem statement

- **Every** module needs to expose configuration
  - HTTP port, JDBC connection pool size, security realm, access log, security permission, SSL certificate, …
- Having every module define its own mechanism is a disaster
  - One configuration file per one module!?
  - Lots of system properties!?
- The whole thing will need to look coherent to users
  - We need domain.xml --- single file that configures everything

One example of this problem

- Extensions only get 2nd class treatment

```
<lifecycle-module
 class-name="com.sun.jbi.SunASJBIBootstrap"
 classpath="../jbi/lib/jbi_framework.jar"
 enabled="true" is-failure-fatal="false"
 name="JBIFramework" object-type="system-all">
     <description>...</description>
     <property name="com.sun.jbi.home"
               value="${com.sun.aas.installRoot}/jbi"/>
     <property name="com.sun.jbi.defaultLogLevel"
               value="WARNING"/>
</lifecycle-module>
```

One example of this problem

- It needs to look like this

```
<jbi home="${com.sun.aas.installRoot}/jbi"
     default-log-level="WARNING" />
```

Config beans don't work

- Because it makes closed-world assumption
  - A 3rd party module developer can't expose his configuration in domain.xml
  - Exposing config means changing DTD and recompiling it
- Because it forces us to do things eagerly
  - To interpret values in config, we need to load respective modules
  - e.g., we need to open TCP port for IIOP right away, but without loading any IIOP code
- Just doesn't work very well with dependency injection

But above all...

- Because config beans need to go up the meta ladder

Making something work for everyone

=

No inherent knowledge about anyone

So instead…

- In essence, we'd like to do it like Spring does
  - But with the power of Java5 language features
- Main ideas
  - Read domain.xml as a blueprint to build component graph
  - Components should behave like real objects
    - That is, state encapsulated by behaviors
  - Have HK2 inject configurations to components
  - Have HK2 do the whole thing as lazily as possible
  - If necessary provide the config beans as compatibility layer

#1 Define components

- HTTP Listener

```
@Configured class HttpListener {
  @FromAttribute void setPort(int) {...}
  @FromAttribute void setVirtualHost(VirtualHost) {...}
}
```

- Virtual Host

```
@Configured class VirtualHost {
  @FromElement void setDocRoot(File f) {...}
}
```

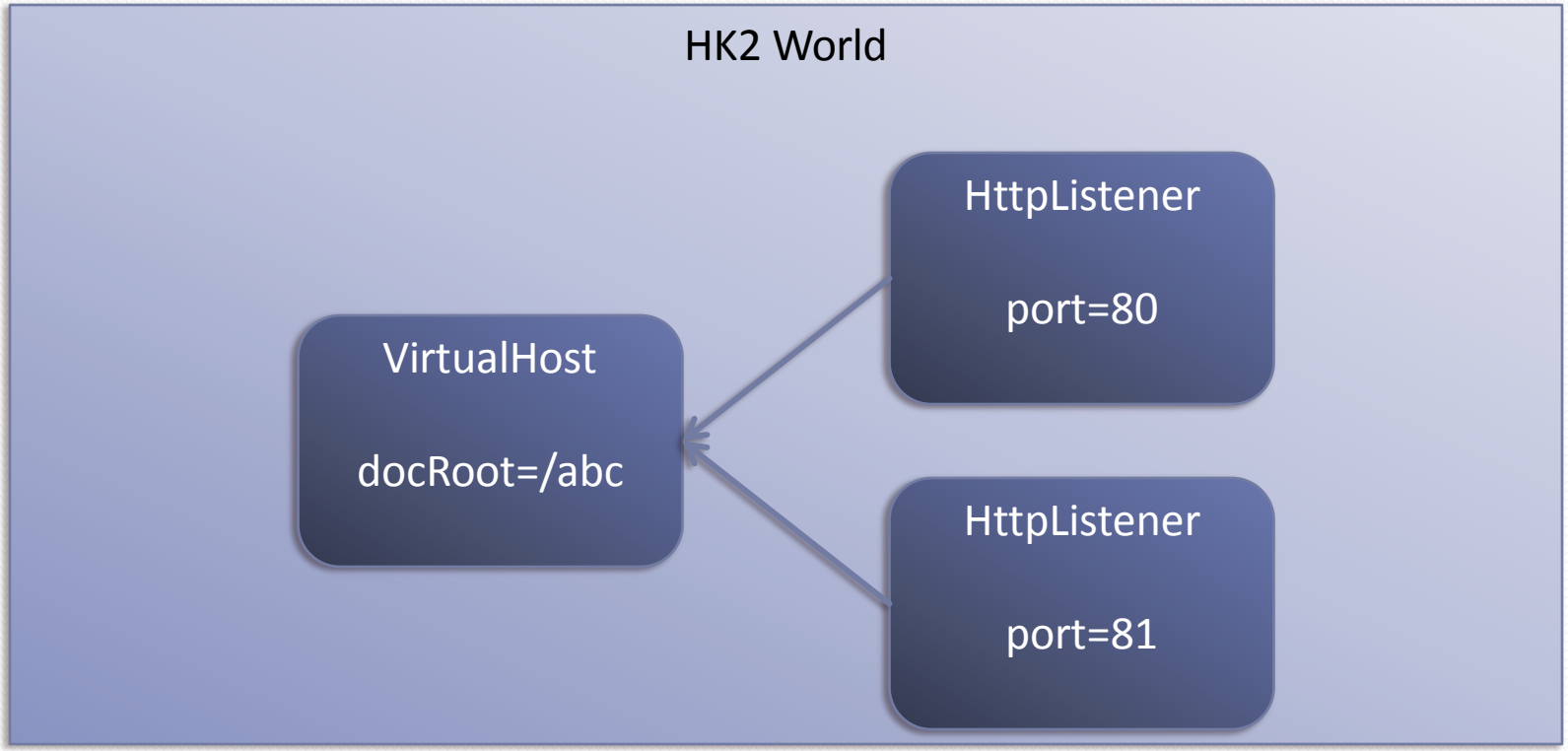
#2 domain.xml

- Annotations determine the shape of XML

```
<domain>
  <virtual-host id="main">
    <doc-root>/abc</doc-root>
  </virtual-host>
  <http-listener port="80" virtual-host="main"/>
  <http-listener port="81" virtual-host="main"/>
</domain>
```

#3 Runtime does the magic

- It will create 3 objects, inject config, and set up references



#4 Reflexive domain.xml access

- HK2 lets you access domain.xml as mini-DOM
  - Used by admin GUI/CUI, AMX, etc. for accessing values
  - Support write back to domain.xml
    - A good library needed to preserve whitespace & comments
- DOM also used for order-insensitive variable expansion

```
<doc-root>${com.sun.aas.instanceRoot}/htdocs</doc-root>
```

- Bi-directional access between DOM and actual components
- DOM changes can be pushed to live objects
  - Components responsible for reacting to changes

Benefits

- Flexible enough to remain compatible with domain.xml
- Promotes good component design
  - Self-sufficient, encapsulated-by-behavior objects
- Eliminates boiler-plate code
  - Configuration is delivered to you, in the right type
  - Much of validation and error diagnostics now done by HK2
- Exposing more configuration is easier & de-centralized
- Module boundary not visible in domain.xml

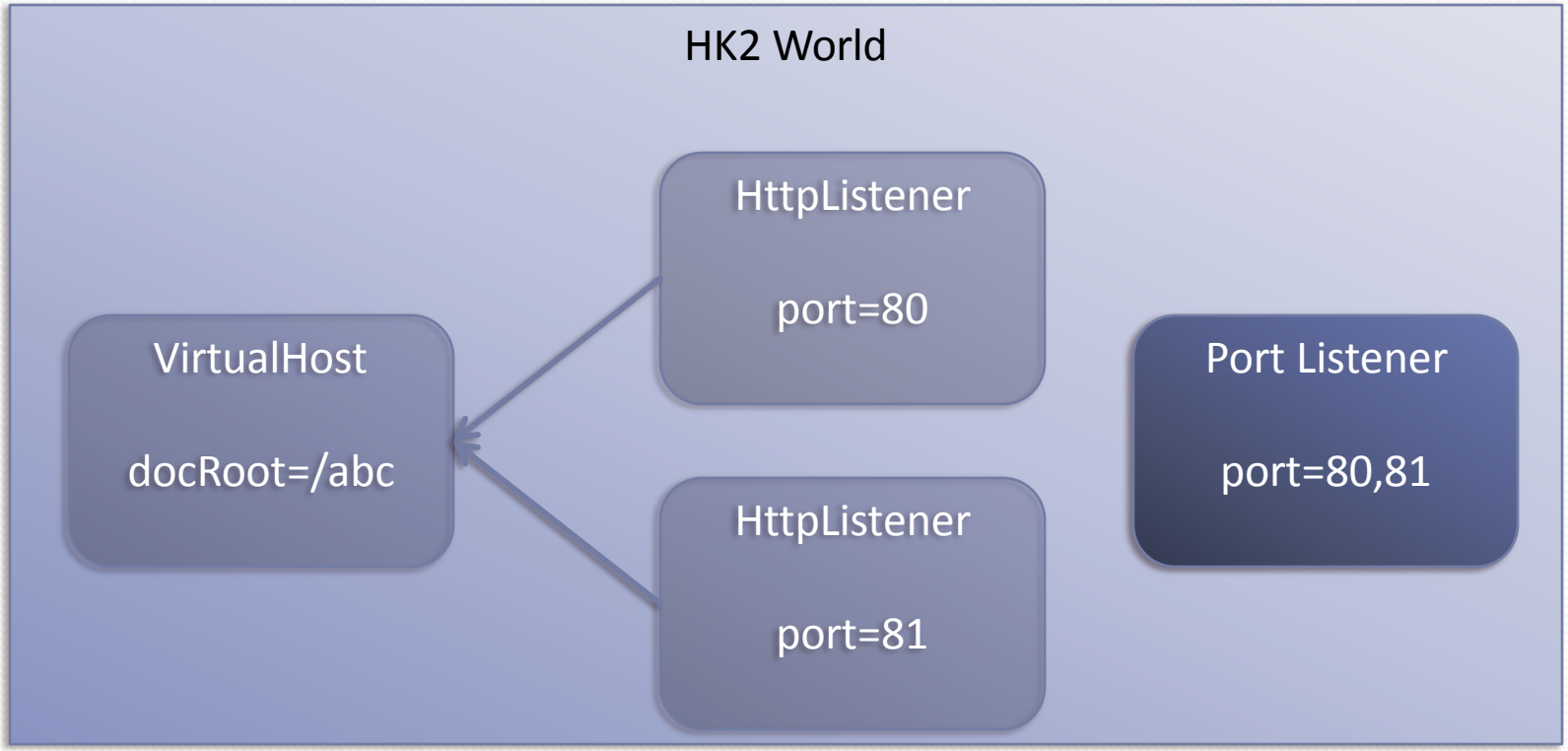
But wait! There's more!

Boot performance

- I expect this to be faster than what we have today
  - Validation now done by DOM impl
    - Validation by DTD/schema etc is an overkill
  - JAXB/schema2beans is slower than W3C DOM
  - This should be faster than W3C DOM

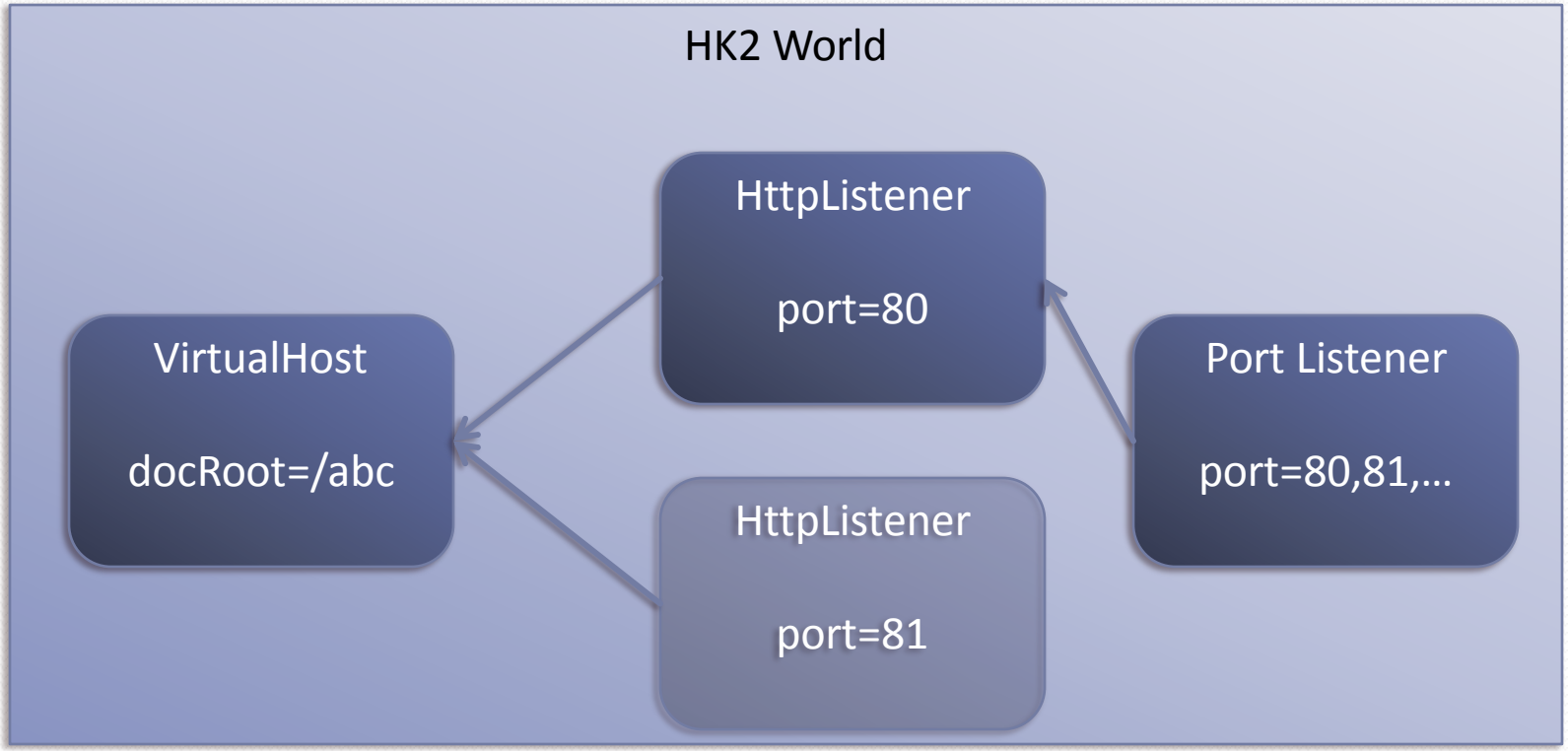
Lazy processing

- Initially port listener will listen to all ports
  - Think of it as inetd



Lazy processing

- PortListener requests an actual object from “the world”, which triggers activation



Schema generation

- At development time
  - Given a distribution POM, we can generate the schema for domain.xml
- At runtime
  - Given the modules that are running, we can generate the schema
- We can also generate HTML documentations
  - Description of configuration now belongs to the code, where it should be.

Dynamic reconfiguration

- Configuration can be changed at runtime
  - Triggers re-injection to reflect changes to running program
  - Change done at infoset-level

```
dom.attribute("docroot"," ${installRoot}/htdocs")
```

  - Can also consider writing code generator that provides statically-typed access
    - java.lang.reflect.Proxy based
    - Note the datatypes are always string because of variables

```
interface VirtualHostDom {
  @Attribute("docroot")
  void docroot(String value);
}
```

Making it practical

Existing code to be affected

- Everyone reading values from domain.xml
  - But some of the cost shall be absorbed into m12n cost
  - More about compatibility layer next
- AMX implementation
- Admin CLI/GUI
  - Anissa told me that they interact with domain.xml through AMX, so maybe it's OK

Compatibility layer

- If the change to the existing code is too much, we need a compatibility layer that feels like old config beans
- Approach
  - Have JAXB generate beans once, commit the source to CVS
  - Replace JAXB annotations by HK2 config annotations

Characteristics

- Cheap and easy. We already do JAXB code generation from domain.xml DTD
- Compatibility config bean can be replaced by real component one element at a time
  - No need for coordinated upgrade
- New modules can take advantage of the new configuration mechanism

Beyond HK2 config

Truly extensible configuration takes more than HK2

AMX

- 3rd party modules need to be able to expose their config through AMX
  - That means we can't be manually writing com.sun.appserv.management.config interfaces
  - We need more dynamic runtime + code generator
- IOW, AMX needs to go up the "meta" ladder, too

Admin GUI

- Every modules must be able to expose its configuration via GUI, too
  - Admin GUI needs to go up the “meta” ladder too
- Some vague ideas, but need serious design
  - Set of annotation/XML/what-not to let module developers contribute GUI