Building without the closed-world assumption

Integrating enterprise build artifacts using Scala

Christos KK Loverdos
loverdos@gmail.com
twitter.com/loverdos
ckkloverdos.com

Who Am I?

- Research-inclined software engineer
- Member of the core team the led the software design & implementation of www.athens2004.com
- Co-author of forthcoming "Steps in Scala"
- Ex Java enthousiast

What this talk is about

- Vendor lock-in frustration
- Using Scala to a pragmatic problem and showing how it was done
- Build artifacts as independent, first-class citizens
- Seeking and proving added value as a means to convince managers/the boss

Closed-world assumption?

Closed-world assumption?

Anyone?

Closed-world assumption?

Anyone?

- http://en.wikipedia.org/wiki/Closed_world_assumption
  - what is not currently known to be true, is false

Closed-world assumption enterprise life

- Vendor lock-in
  - "what is not controlled by Vendor does not take the proper treatment" - if any

The setting

General Facts

- Brand-new J2EE project
  - First major use of Java > 1.4 for customer
- New set of tools
  - IDE that promise the best programming experience
- Junior developers team
  - Enthusiasm but lack of experience

The setting

Problems begin

- Fancy IDE but just for the unexperienced
- Easy to work with Vendor extensions but not with standards-compliant components
- One-click-away software generation not very helpful in educating the younger

The setting

Problems continued

- Built-in support for other tools outdated
  - Could not install latest maven plugin
- Exported ant scripts meant to be untouched
- We had the core framework developed using maven
- Exported maven scripts looked like a mess

Preliminary thoughts

- One-click semantics considered harmful
- Desperately need some automated build system
- Build artifacts must be integrated
- Break away from Vendor lock-in

Vendor lock-in is bad

- Having no alternatives can be catastrophic
- Can do harm to standards adoption
- Reduces creativity
- ...

Answer to lock-in

Answer to lock-in

Why Scala?

- Convince managers that a better tool was necessary
  - Productivity boost
  - Knowledge horizon expansion

Terminology & design

- A build artifact is a file, generated by some build procedure
- A build component is something that needs to be built
  - EJB, Web Service, Domain Model, ...

Terminology & design

- Build components depend on one another
- Nested build components
  - Just for one level in the initial impl.

Terminology & design

- A builder type is a mechanism that drives a build process
  - ant, maven, vendor-ant, ...
- Default build actions
  - clean, compile, build, clean-build

Terminology & design

- Automatic discovery of builder type
  - Predefined rules in the initial impl.
  - Easy to refactor to a strategy

Builder type discovery

```
def discoverBuilderType(place: RichFile): BuilderType = {
   val antBuildXml    = place / "build.xml"
   val mvnPomXml      = place / "pom.xml"
   val vendorBuildXml = place / "vendor-build.xml"
   if(mvnPomXml.exists)
      MavenBuilderType
   else if(antBuildXml.exists)
      if(vendorBuildXml.exists)
         VendorAntBuilderType
      else
         AntBuilderType
   else
      NoBuilderType
}
```

Terminology & design

- Declarative properties per build component
  - generated-artifacts
  - depends-on
  - enable
  - java-classpath

A few library goodies

- RichFiles with better path handling
  - componentHome / "build.xml"

A few library goodies

- Immutable FileSets
  - case class FileSet(files: List[RichFile])
    - ... with the ability to be filtered
      - def *(filter: NameFilter) =
        - FileSet(files.filter(filter.accept(_.name)))
    - ... using glob patterns

A few library goodies

```
trait NameFilter {
   def accept(name: String): Boolean

   def |(other: NameFilter): NameFilter = ...
   def unary_!: NameFilter = ...
}

class GlobFilter(glob: String) extends NameFilter {
   val globRE = globToRegexStr(glob)

   def accept(name: String) = name.matches(globRE)
}
```

Results

- Here comes the fun part...

Universal Build & Deploy

```
+-------------------------------------------+
|  1. Build Core Framework                  |
|  2. Build other component                 |
|  3. Call specific action                  |
|  4. Show component types                  |
+-------------------------------------------+
|  5. Show auto build order                 |
|  6. Build all [based on auto build order] |
|  7. Make dependency graph  [via graphviz] |
+-------------------------------------------+
|  8. Show generated artifacts              |
|  9. Clean generated artifacts             |
+-------------------------------------------+
| 10. Deploy                                |
| 11. Undeploy                              |
| 12. Redeploy                              |
+-------------------------------------------+
| 13. Exit                                  |
+-------------------------------------------+
Enter Choice:
```

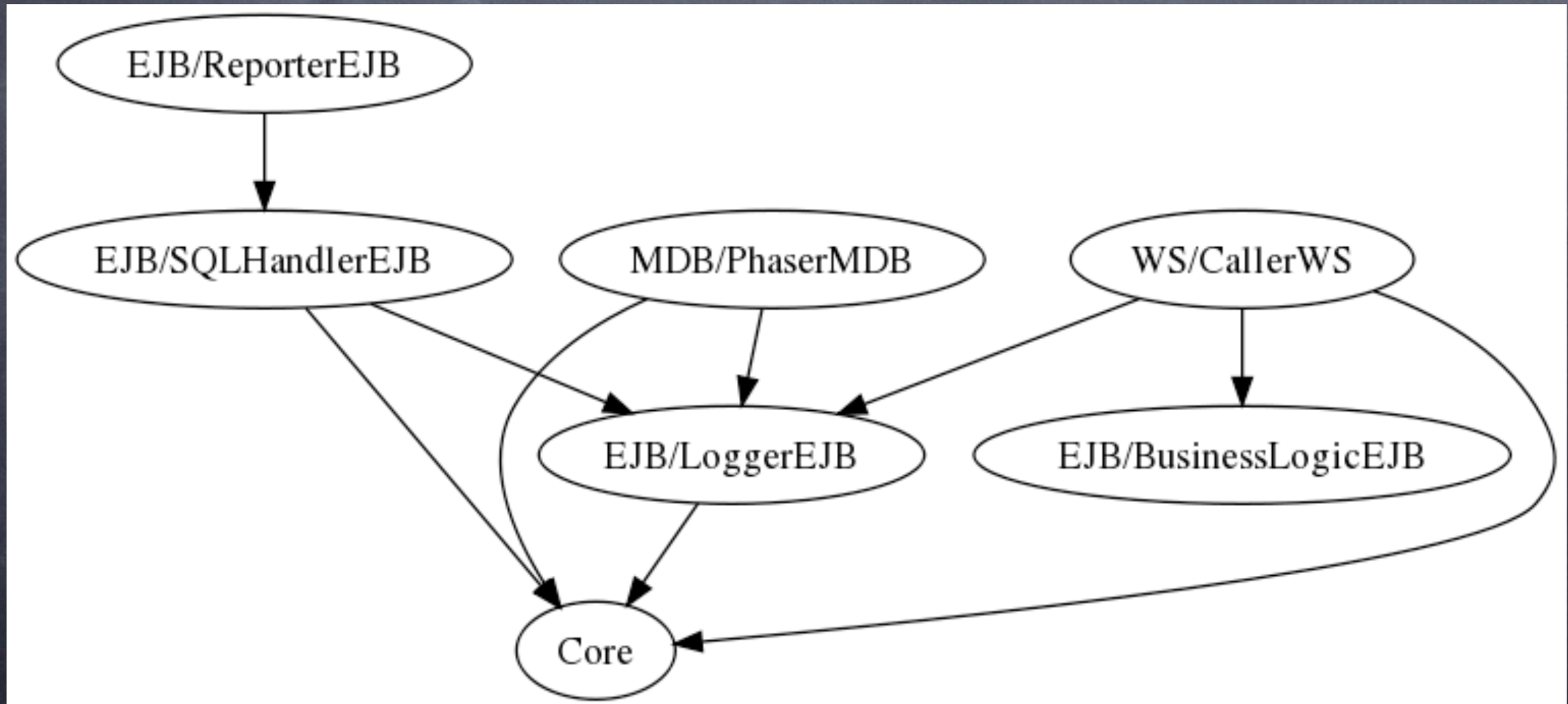
Our Build Components

```
[    mvn] Core
[vnd-ant] WS/CallerWS
[    ant] EJB/BusinessLogicEJB
[    ant] EJB/LoggerEJB
[    ant] EJB/SQLHandlerEJB
[    ant] MDB/PhaserMDB
[    ant] MDB/ReporterMDB

=========================================
Last command was: 4. Show component types
=========================================
```

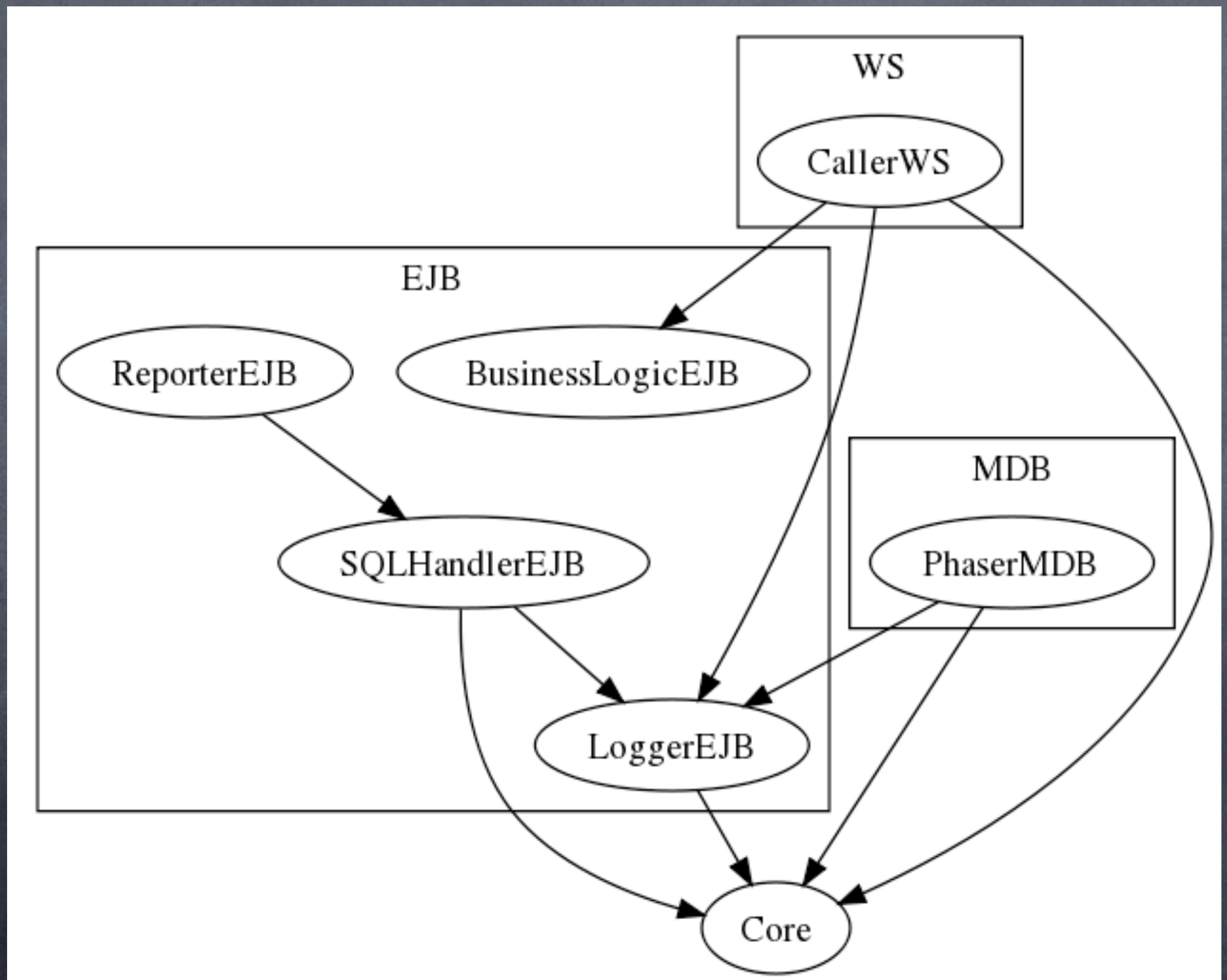
Automatic dependencies

```
First    => Core
 Then    => EJB/LoggerEJB
 Then    => MDB/PhaserMDB
 Then    => EJB/BusinessLogicEJB
 Then    => EJB/SQLHandlerEJB
 Then    => EJB/ReporterEJB
Finally => WS/CallerWS

=========================================
Last command was: 5. Show auto build order
=========================================
```

Also graphically



...and clustered



Deployable artifacts

```
Generated: [2010-01-01 15:15:15] Core-1.1.jar
Generated: [2010-01-01 15:15:17] EJB/LoggerEJB.jar
Generated: [2010-01-01 15:15:20] MDB/PhaserMDB.jar
Generated: [2010-01-01 15:15:23] EJB/BusinessLogicEJB.jar
Generated: [2010-01-01 15:15:24] EJB/SQLHandlerEJB.jar
Generated: [2010-01-01 15:15:26] EJB/ReporterEJB.jar
Generated: [2010-01-01 15:15:39] WS/CallerWS.war

================================================================
Last command was: 8. Show generated artifacts
================================================================
```

...and how they are built

```
[OK] [2010-01-01 15:15:15] Core-1.1.jar          for Core
[OK] [2010-01-01 15:15:17] LoggerEJB.jar         for EJB/LoggerEJB
[OK] [2010-01-01 15:15:20] PhaserMDB.jar         for MDB/PhaserMDB
[OK] [2010-01-01 15:15:23] BusinessLogicEJB.jar for EJB/Business...
[OK] [2010-01-01 15:15:24] SQLHandlerEJB.jar     for EJB/SQLHandl...
[OK] [2010-01-01 15:15:26] ReporterEJB.jar       for EJB/ReporterEJB
[OK] [2010-01-01 15:15:39] CallerWS.war          for WS/CallerWS

================================================================================
Last command was: 6. Build all [based on auto build order]
================================================================================
```

Thank you!