ThoughtWorks®
The art of heavy lifting.℠

Constructor Dependency Injection

A post J2EE Nirvana

V1.1 © Paul Hammant, ThoughtWorks Inc.
Paul@ThoughtWorks.com

O'Reilly Open Source Convention 2004 in Portland, Oregon
Date: Friday, July 30 ; Time: 10:45am - 11:30am
Track: Java, Location: Salon H

J2EE is harder than it should be*

- Too much JNDI.
- Complicated separations between beans and remotes and homes.
- RemoteException or not.
- No simple Unit testing like plain classes.
- It feels cumbersome and is not transparent
- Too much XML meta-data
- By no means simple

* Addressed *in part* by the recent EJB 3 draft specification

Transparency and Constructors* are the way to go

* For dependency resolution

Transparency in words

- Domain model in style of service layer pattern (façade-like interfaces)
- Client logic uses the service layer model
  - mockable for unit testing
  - usable outside of container
- Implementation of domain model utilizes hierarchies of components to implement interfaces.

Transparency in code

```
interface Account {
  void deposit(Money amt);
  // other method declarations
}

class CurrentAccount implements Account {
  // method implementations
}
```

Note - no extends, implements or throws from a framework.

Dependencies via 'new' is wrong*

```
class CurrentAccount implements Account {
  final PersistenceStore store;
  public CurrentAccount () {
   store = new PersistenceStoreImpl();
  }
  // more business methods needing store ...
}
```

Components should not resolve their dependencies via instantiation. This hinders unit-testing.

* Likely to cause defensive reaction

Setters* lead to more meta-data

```
class CurrentAccount implements Account {
  PersistenceStore pStore;
  public void setStore(PersistenceStore ps) {
    this.pStore = ps;
  }
  // more business methods needing store ...
}
```

- How many setters to set post instantiation?
- Which setters are mandatory?
- Likely to require XML Meta-data

* For dependency resolution

Service Locator* entangles things

```
class CurrentAccount implements Account {
 public void someMethod () {
   PersistenceStore store=ServiceLocator.getStore();
     store.doSomething(true);
 }
 // more business methods needing store ...
}
```

- Component entanglement
- Dependencies are hidden
- Unit testing pitfalls
- Hard to manage comps with restricted scope
- Raymen Noodle / Hairballs can emerge

* For dependency resolution

Constructor* is best

```
class CurrentAccount implements Account {
  final PersistenceStore store;
  public CurrentAccount(PersistenceStore store) {
      this.store = store;
  }
  // more business methods needing store ...
}
```

Constructor Dependency Injection:-

- Simple & fail fast
- Declarative without metadata
- Usable **without container**
- Relevant to all OO languages

* For dependency resolution

"I was expecting
a **paradigm shift**,
and all I got was a
lousy constructor!"

Introducing PicoContainer

- Register components by type and implementation
- Neatly handles dependency resolution and injection before instantiation
- PicoContainer can handle lifecycle concepts such as start() and stop() of daemon or thread-like components
- Pure library / small jar

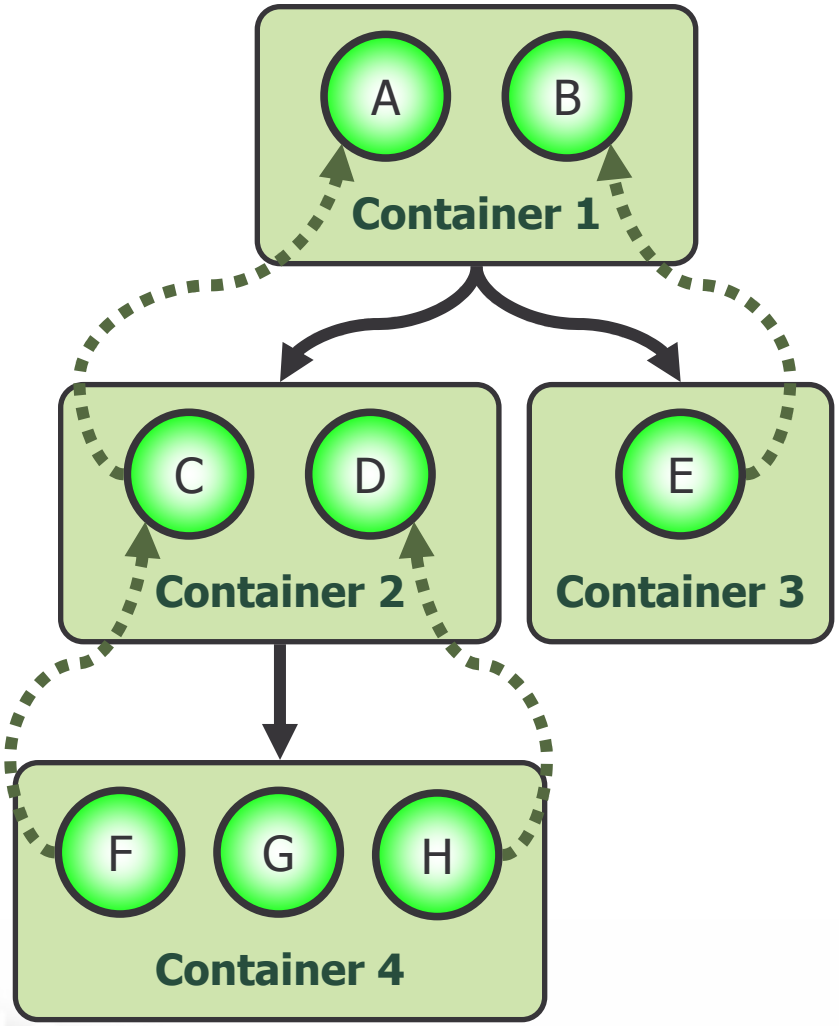
- Most useful when custom soft composition of components is needed or when handling multiple implementation of a type.

PicoContainer example

```
// assume accountType (of type Class) passed into method..

MutablePicoContainer pico = new DefaultPicoContainer();
pico.registerInstance(PersistenceStore.class, myStore);
pico.registerImplementation(SuperCurrencyConverter.class);
pico.registerImplementation(FraudCheckerImpl.class);
pico.registerImplementation(Account.class, accountType);

Account ca = (Account)  pico.getComponentInstance(Account.class);

ca.deposit(new Money());
```

CurrentAccount only needs PersistenceStore, but other Account types could need FraudChecker and/or CurrencyConverter. PicoContainer handles it.

Container hierarchies aid separations



- Allows scoping of comps. e.g. Component 'D' can't depend on, or use 'E'
- Application, Session, and Request scope (servlets) is one sweet spot

Lifecycle management cascades from parent to child containers if applicable

Dependencies are resolved from child thru parents (back to root if necessary)

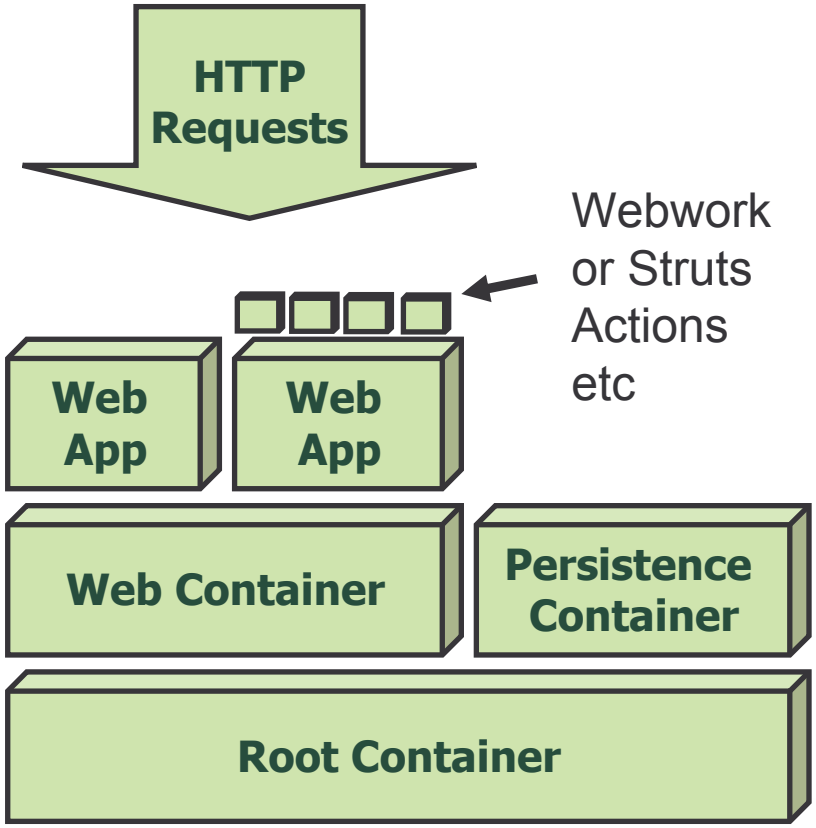
NanoContainer

- Adds formal scriptable composition to PicoContainer via Groovy, XML etc
- Also adds classloader hierarchies allowing restrictive security (even sandboxing)
- Servlet web-apps use is a sweet spot
  - enablers for Struts, Webwork in 'NanoWar'
  - If used with 'Jervlet', even servlets are CDI enabled.

Groovy Builder example

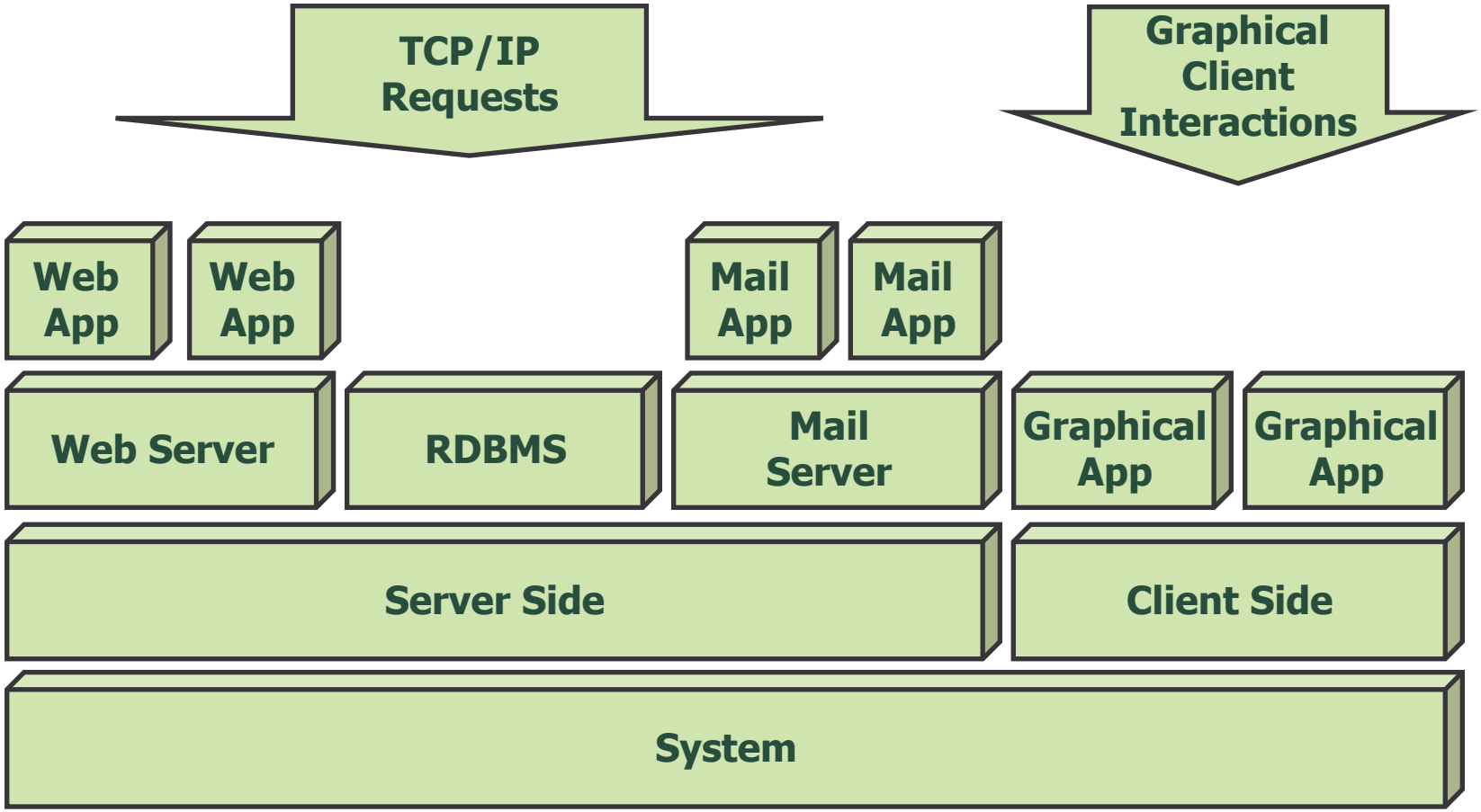
```
• builder = new NanoGroovyBuilder()
• pico = builder.container() {
•   classpathelement(path:"lib/foo.jar");
•   component(key:Foo, class:"DefaultFoo")
•   component(key:Bar, class:"DefaultBar")
•   component(class:"MyBeanShellConsole")
•   container() {
•     component(class:Huey)
•   }
•   container() {
•     component(class:Duey)
•   }
• }
```

MicroContainer



- Compose entire applications using plugins (.mca files)
- Plugins are NanoContainer scripts
- Standalone or Embeddable
- (alpha)

Component Nirvana



One day whole Operating systems will be composed via CDI, perhaps even traversing language barriers.

The Gang

Paul Hammant – co lead

(Jesktop, Enterprise Object Broker, AltRMI)

Aslak Hellesøy – co lead, pictured

(XDoclet, MiddleGen, DamageControl)

Jon Tirsen – prolific hacker, pictured

(Nanning, DamageControl, Prevayler)

Joe Walnes – pictured

(QDox, Sitemesh, NMock, JMock, XStream)

Also - Chris Stevenson, Dan North, Mike Hogan, Konstantin Pribluda, Jörg Schaible, Thomas Heller, Leo Simmons, James Strachan, Maartin Grootendorst, Stephen Molitor, Mike Ward, Mauro Talevi.

References

- http://paulhammant.com/downloads/PicoNirvana.pdf
- - This slide show

- http://www.martinfowler.com/articles/injection.html
  - Martin Fowler's article on the family of patterns.

- http://picocontainer.org http://nanocontainer.org http://microcontainer.org

- http://picocontainer.org/Constructor+Injection
  – PicoContainer's definition of Constructor Dependency Injection

Notes

- PicoContainer has Ruby, .Net and PHP ports too
  – check out the web site.
- We'd really love people to help out with a Python port. Is there need for a Perl port?