OSGi

Building LinkedIn's Next Generation Architecture with OSGI

Yan Pujante
Distinguished Software Engineer
Member of the Founding Team @ LinkedIn
ypujante@linkedin.com
http://www.linkedin.com/in/yan
Background

- LinkedIn created in 03/2003
  - close to 30M members
  - profitable with 5 revenue lines
  - HQ in Mountain View (400 employees)

- Technologies
  - 2 datacenters (~ 600 machines)
  - SOA, java, tomcat, spring framework, http, Oracle, MySQL, servlets, jsp, graph, OSGi...
  - dev on MacOS X, prod on Solaris
Challenges

- Growing engineering team on a monolithic code base (still modular)
- Growing product team wanting more and more feature, faster
- Growing operations team deploying more and more servers
Challenges (SOA in dev)

- **Front-end:** many BL services in one webapp (in tomcat)
- **Back-end:** many wars in 5 containers (jetty) with 1 service per war
- Note that prod and dev are different
Container Challenge

1 war with N services

- does not scale for dev (conflicts, monolithic...)

N wars with 1 service

- does not scale from a container point of view (no shared jars)
  - Can add containers but only 12Gb of RAM :(
Challenges (SOA in prod)

- Upgrading back-end service to new version requires downtime
  - Hardware LB does not account for version

- Upgrading front-end service to new version requires redeploy
  - N services in 1 war

- Adding new back-end service painful
  - lots of manual configuration (LB, front-end...)
Solution: OSGi

- Belief that OSGi is one of the tools that will help us address those challenges

Why OSGi?
OSGi intro

- OSGi means “Open Services Gateway initiative”
- OSGi is a spec with several implementations
  - Equinox (eclipse!)
  - Knoplerfish
  - Felix (Apache)
OSGi intro (cont.)

- OSGi has some very good properties
  - Smart class loading => multiple versions jars ok
  - Highly dynamic => deploy / undeploy built in
  - Services registry
  - Highly extensible / configurable

- OSGi bundle = Jar File with OSGi manifests
Specification-Title: Colorado Software Summit 2008 Hello World Client

Bundle-Activator:
  com.linkedin.colorado.helloworld.client.HelloWorldClientActivator

Import-Package:
  com.linkedin.colorado.helloworld.api;version="[1.0.0,1.0.1)",
  com.linkedin.colorado.helloworld.client;version="[1.0.0,1.0.1)",
  org.osgi.framework;version="[1.4.0,1.4.1)"

Export-Package:
  com.linkedin.colorado.helloworld.client;version="1.0.0"

Bundle-Version: 1.0.0

Bundle-Name: colorado-helloworld-client

Bundle-ManifestVersion: 2

Bundle-SymbolicName: colorado-helloworld-client

Bnd-LastModified: 1224557973403

Generated-On: 20081020

Tool: Bnd-<unknown version>

Implementation-Version: DevBuild
BundleActivator

- In a war: ServletContextListener

```java
public interface ServletContextListener extends EventListener {
    void contextInitialized(ServletContextEvent event);
    void contextDestroyed(ServletContextEvent event);
}
```

```xml
web.xml
    <listener>
        <listener-class>com.colorado.MyListener</listener-class>
    </listener>
```

- In a Bundle: BundleActivator

```java
public interface BundleActivator {
    void start(BundleContext bundleContext) throws Exception;
    void stop(BundleContext bundleContext) throws Exception;
}
```

```text
MANIFEST.MF
Bundle-Activator: com.colorado.MyActivator
```
BundleContext

- The bundle context is the entry point:
  - registerService / getServiceReference => to register / get services from registry
  - addBundleListener / addServiceListener => to listens / extends the framework
  - even installBundle!
  - ... and more
Demo

- Equinox demo... stay tuned...
Web Container

- Services exported with Spring/RPC
- Services in same war can see each other
- Share the jars
Web Container

- Cannot share jars
- Cannot talk directly
OSGi container

- Bundles (jars) shared
- Services shared
- Dynamic replacement of bundles
  ➢ OSGi solves the container challenge
Distributed OSGi

- OSGi is currently container centric
- Next version (4.2) will introduce distributed OSGi

➢ What do we do in the meantime?
LinkedIn distributed OSGi

- Service ‘red’ talks to service ‘green’ (local)
  - What happens if we move ‘green’ to a different container?
LinkedIn distributed OSGi
LinkedIn distributed OSGi
LinkedIn distributed OSGi

Bundles

Service Registry

C1 VM

C2 VM

Bundles

Service Registry

C3 VM

Yan Pujante — OSGi: Building LinkedIn’s Next Generation Architecture with OSGI
LinkedIn distributed OSGi

- no more N-1 / 1-N problem
  - libraries and services can be shared in 1 container (mem. footprint smaller)
  - services shared across containers
- transparent location to clients
  - no more configurations to change when adding/removing/moving services
- software load balancer
  - much more flexible (handle version)
LB and version
Everything is perfect...

- ...Not really
  - OSGi has some problems
OSGi problems

- OSGi is great, but tooling is not quite there yet
- Not every lib is a bundle
- OSGi was designed for embedded devices => using it for server side is recent (but very active)
OSGi problems

- OSGi is quite low level => but there is some work to ‘hide’ the complexity
  - spring DM (spring dynamic module)
  - multiple vendors containers
    - Spring DM Server, Glassfish, Infiniflow

- OSGi is container centric, but next version will add distributed OSGi (although no support for LB)
OSGi problems (cont.)

- version management can be nightmarish
  - Import-Package:
    
    ```
    com.linkedin.util;version=1.0.0
    ```
  - version=1.0.0 means \([1.0.0, \infty)\)

- should use at least:
  - version=\([1.0.0,2.0.0)\)

- runtime is dynamic and can result in untested code being executed!
Runtime version problem

- client1 and client2 uses Service 3.0.0
- Dependencies are ‘loose’: [3.0.0, 4.0.0)
We upgrade client1 and service

client2 starts using green bundles

=> Most likely untested
Runtime version solutions?

- A solution is to lock version:
  - $[1.0.0, 1.0.0] \Rightarrow \text{no slack}$
  - $[1.0.0, 1.0.1) \Rightarrow \text{tiny bit of slack for emergency mode}$

- Does it really work? spring 2.5.5 imports hibernate version $[3.1, \infty)$

- Can OSGi even resolve with everything locked down?
Runtime version problem

- With maximal lock-down, client2 cannot use the new service
Services cannot talk to each other directly
=> need serialization for backward compatible services (dynamic proxy)
Despite problems...

- OSGi and community moving in the right direction
Where are we?

- Bundle repository (big amount of work... in progress)
- Use ivy/bnd for generating bundles
- Evaluating container
  - Infiniflow (from Paremus) most likely the one
  - LinkedIn spring / SCA to deploy composites
- Work on the load balancer/distribution (in progress)
- Work on tooling / build integration (Sigil from Paremus)
- Much much more to do...
Conclusion

- Using OSGi... definitely!
- But we do not want application developers to have to deal with it.
More information

- Check my posts on the LinkedIn blog (OSGi series). More to come...