eclipseCON™ 2008

How To Build Large Scale Enterprise Applications Using OSGi

Who Am I?

- 10 Years as a developer and architect
- Mainly startups in telecoms and finance
- Whole range of software lifecycle, design, dev, support and admin

What Have I Been Working On?

- Provisioning
- Installation
- Remote Classloading in OSGi
- Eclipse Plugin

Who Are Paremus?

- UK based software company
- Started as a consultancy in 1999
- Formed by senior architects and decision makers
- Switched to software company in 2004
- Not Grid, Not N Tier, Fabric

Large Scale Enterprise Infrastructure

- 100+ Core’s
- Multiple Applications
- Multiple Requirements
- Multiple Users

Types Of Industry

- Biotech
- Engineering
- Finance
- Games
- Military
- Web 2.0/3.0?

Newton. infiniflow

- Open Source Framework/Commercial Application
- Version 1.2 GA: March 2008
- Based on OSGi
- Deploy/Manage OSGi applications life-cycle on remote nodes

Newton @ Citigroup

- London Development Center
- 800 Cores (8 CPU Boxes 16 GB each)
- 100 Newton Instances
- 800 User Components

Focus Of Talk

- Discussion vs Lecture
- Not trying to sell you:
  - Any particular remote protocol
  - Any particular programming model
  - Any particular software ;)
- How do we build large scale apps - architecture, sys admin, etc.

What's The Problem?

Why Do We Scale?

- Increased Power
- Increased Throughput
- Reduced Latency
- Increased Resilience

Problems Of Scale

- Complexity
- Reliability
- Inflexibility

An Ideal Enterprise Architecture

Simplicity

- *Make everything as simple as possible, but no simpler* (Quote:Einstein)
- POJOs
  - no dependencies, just business logic
  - make developers lives easier - what about the rest of the enterprise?
- Scale free patterns:
  - Discovery vs static config
  - Characteristics vs Lists

Simplicity

- Enterprise complexity shouldn't get in the way of:
  - the developer,
  - or the architect,
  - or the system administrator
- Make Tasks As Simple As Possible For The Individual

Model Driven

- Need to define the boundary conditions
  - POJOs - need to add meta data somewhere
  - Scale Free: where, when, how to scale
  - Allows you to specify the service level dependencies
- Various DSL's to describe models
- Test deployment scenarios before you commit your infrastructure
- Allows An Enterprise To Quickly Introspect And Change Their Infrastructure Based On Business Requirements

Why Take OSGi Into The Enterprise?

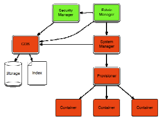
- Imports/Exports
- Simple
- Dynamic
- Security
- Mature
- OSGi is an integral part of a Model Driven Architecture

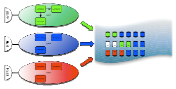
Building Blocks

Newton Architecture

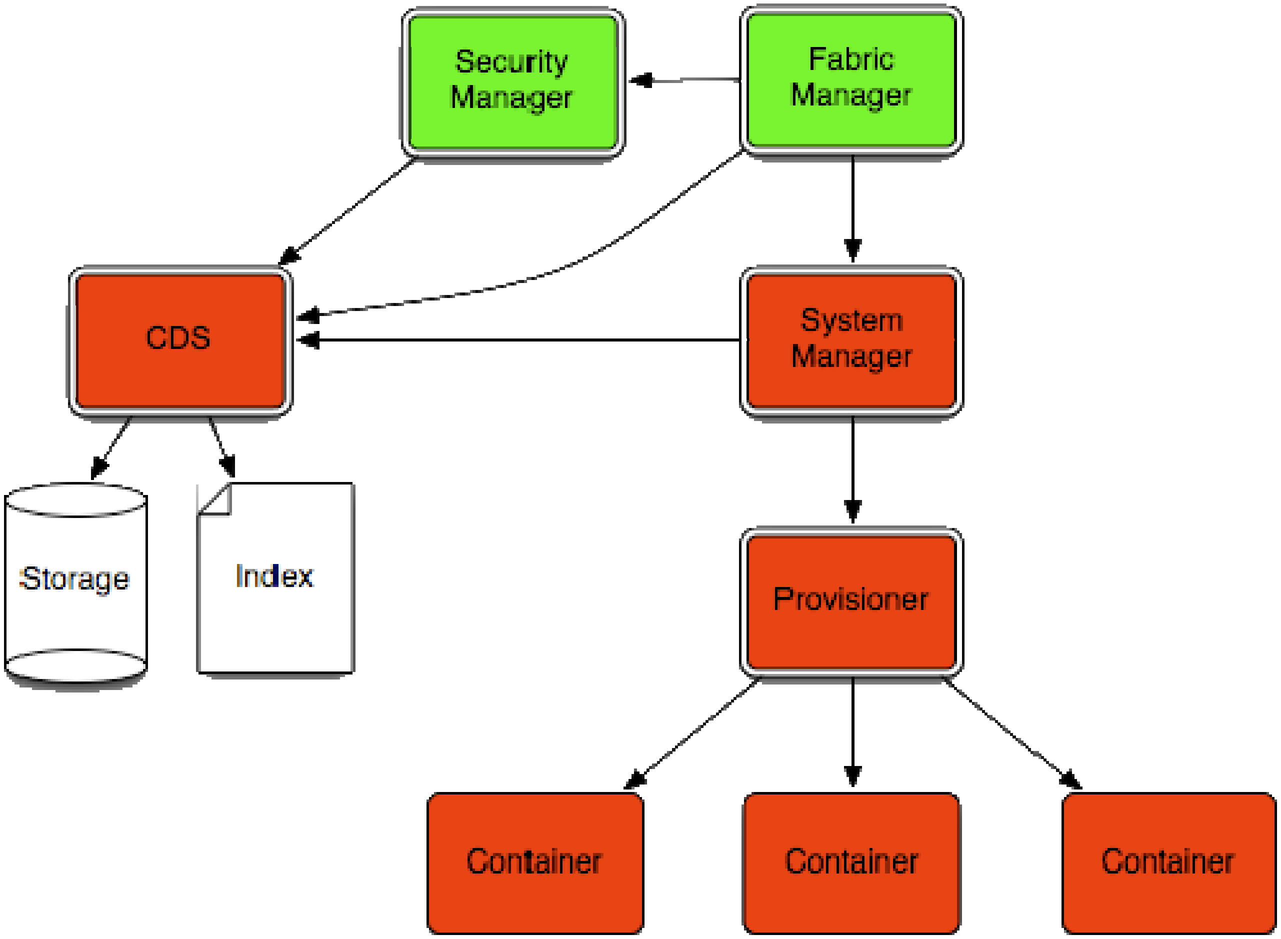
- Components
- Bundles
- Repository
- System Manager
- Provisioner
- Container
- Discovery
- Bindings
- Management

View From 100,000 ft.

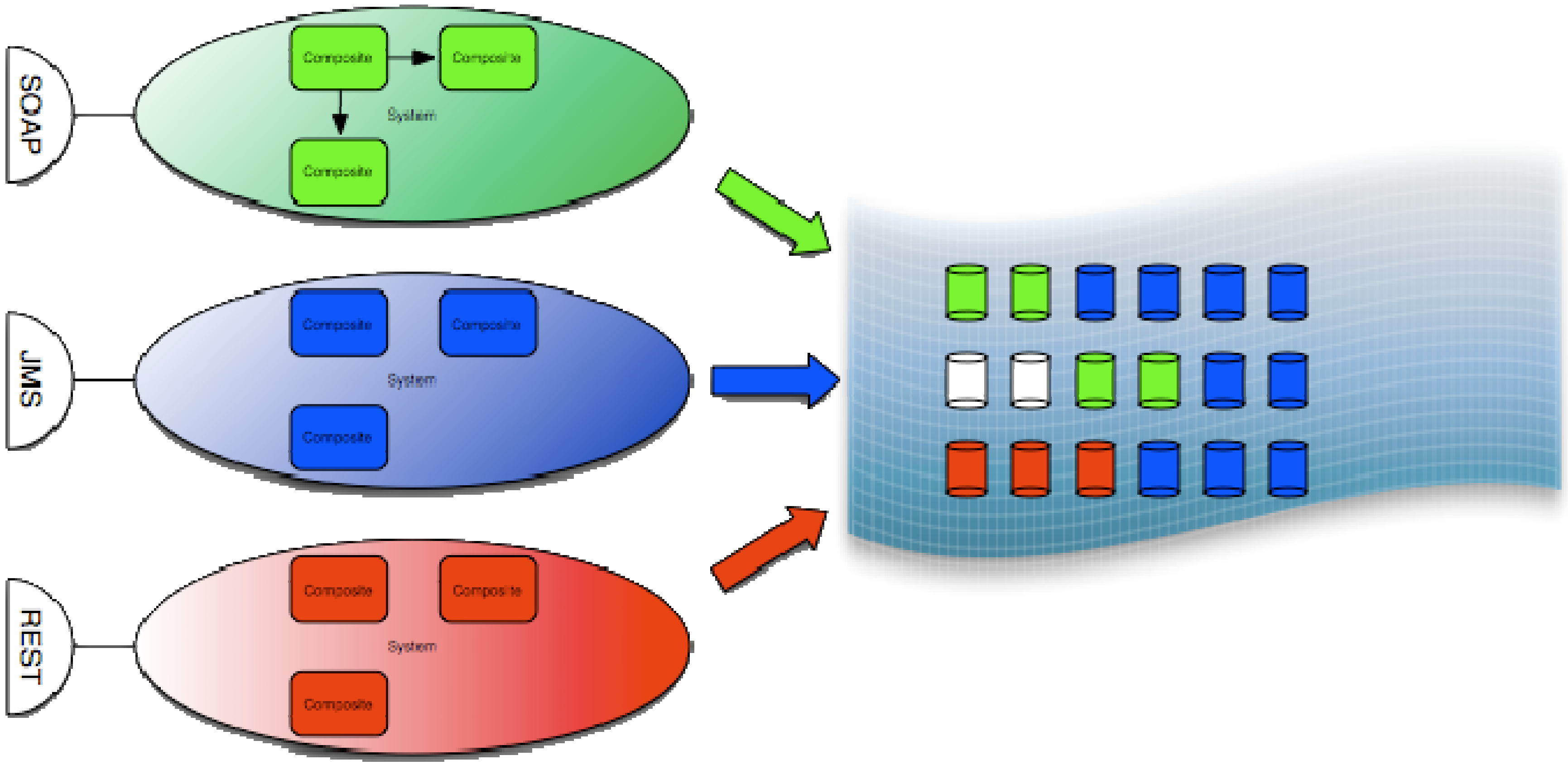




View From 10,000 ft. (Functional Perspective)



View From 10,000 ft. (User Perspective)



Details, details...

Components

- Supported models
  - SCA
  - Spring
  - OSGi
  - Switzerland of component deployment technologies

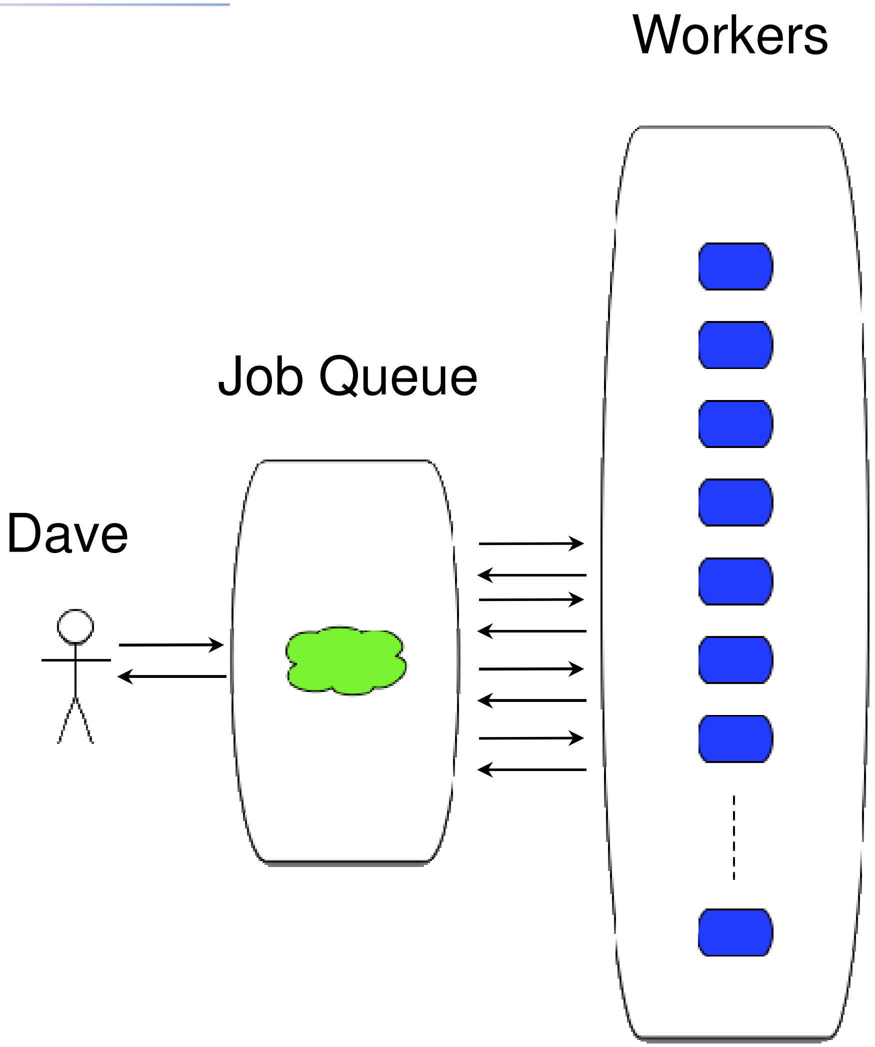
Repository

- Resilient/Highly available
- Query Interface (LDAP)
- Content sensitive

Systems

- Structured group of components
- Defines:
  - which components
  - scaling behaviour
  - configuration

Systems



Systems

```
<?xml version="1.0" encoding="iso-8859-1"?>
<system name="fractal-workers" boundary="fabric">
  <description>An environment that sets up one worker for every container in the fabric</description>
  <system.composite name="worker"
                    bundle="org.cauldron.newton.fractal.engine"
                    template="fractal-worker-template"
                    version="1.0.0">

    <replication.handler name="scale" type="org.cauldron.newton.system.replication.ScalableReplicationHandler">
      <property name="scaleFactor" value="1" type="float" />
      <property name="fixedDelta" value="-1" type="integer" />
      <property name="minimum" value="1" type="integer" />
    </replication.handler>

  </system.composite>

  <system.composite name="space"
                    bundle="org.dancres.blitz"
                    template="blitzTemplate"
                    version="1.2">
  </system.composite>
</system>
```

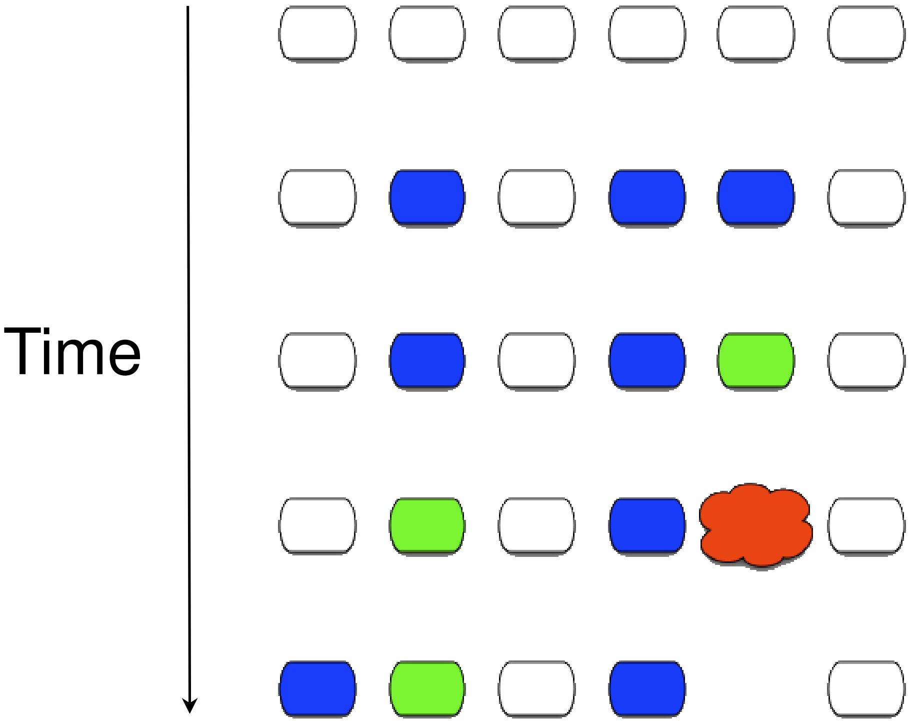
ReplicationHandler

```
public interface ReplicationHandler {
    void initialise(SystemContext ctx) throws SystemConfigurationException;

    SystemContext getSystemContext() throws SystemConfigurationException;

    Collection<CompositeInstance> replicate(
        SystemCompositeDescriptor template,
        SystemStateModel model)
            throws SystemConfigurationException;

    void destroy(SystemContext ctx);
}
```

Presence

- Lightweight pattern for announcing location of components
- Pluggable architecture for defining which components are “activated”

Active Passive Presence Behaviour



ActivePassive

```
<presence-group labels="mylabel"
strategy="org.cauldron.newton.presence.strategy.ActivePassiveStrategy">
  <property name="maximumActive" value="1" type="integer" />
</presence-group>
```

PresenceStrategy

```
public interface PresenceStrategy {
    void configure(Map<String, Object> config);
    long execute(PresenceGroup group);
}
```

Negotiation

- In An Enterprise Environment Not All Nodes Are Equal
  - Need To Make Best Use Of Resource
- Contracts
  - Requirement
  - Cost/Value
  - Assessment
- Features
  - Extensible
- Life-cycle
  - Lifetime
  - Life-cycle events, initialised, bound, unbound, destroy

Contract

```
<contract features="(&amp;(machine.arch=i386)(machine.availableProcessors>1))"/>

<contract features="(!(container.composite.foo>0))"/>

<contract features="(network.bandwidth.in>1M)"/>
```

Assessment

```
public interface CompositeAssessor {
    public static final int MINIMUM_PRIORITY = 1000;
    public static final int MAXIMIM_PRIORITY = 0;
    public static final int NORMAL_PRIORITY = 500;

    int getPriority();

    /**
     * Return a positive integer value representing the "cost" to host the given
     * instance, or -1 if the instance cannot be installed for any cost
     */
    int accept(CompositeInstance instance, Collection<CompositeInstance> current);
}
```

FeaturesProvider

```
public interface FeaturesProvider {
    String getNamespace();

    void addFeaturesChangeListener(FeatureChangeListener listener);

    void removeFeaturesChangeListener(FeatureChangeListener listener);

    Collection<String> getFeatureNames();

    <T> ContainerFeature<T> getFeature(String name);
}
```

Installation And Garbage Collection

- Bundle install - imports/exports/requires/versions
- Component install - services/references - bindings/interfaces
- Graph walker
- Simplifies Job Of System Administrator When Deploying Changes To An Application

Discovery

- Registry
- Query Interface
- Filters and Attributes

Filters and Attributes

```
<reference name="foo">
  <interface.java interface="com.example.Foo"/>
  <binding.rmi filter="(check=true)">
</reference>

<service name="foo">
  <interface.java interface="com.example.Foo"/>
  <binding.rmi>
    <attribute name="check" value="true" />
  </binding.rmi>
</service>
```

SCA Bindings

- OSGi (local)
- RMI
- WS*
- JMS
- Java Space
- ...

Marshalling Java Objects

- Bundle Import/Export not enough
- Private classes are serialized
- Lifecycle of OSGi bundles when unmarshalled

Management

- JMX
- Dynamic
- Pluggable
- Views/Trackers

Tracking MBeans

```
ManagementBeanListener listener = new ManagementBeanListener() {
  public void addBean(ObjectName name, MBeanServerConnection sourceConnection) {
    log.info("Added " + name);
  }

  public void changedBean(ObjectName name, MBeanServerConnection sourceConnection)
{
    log.info("Changed " + name);
  }

  public void removeBean(ObjectName name, MBeanServerConnection sourceConnection) {
    log.info("Removed " + name);
  }
}
```

Tracking MBeans

```
@Reference
private ManagementView view;

...

String CONTAINER =
"org.cauldron.newton:Type=ContainerMBean,*";

ObjectName name = new ObjectName( CONTAINER );

ManagementBeanTracker tracker = new
ManagementBeanTracker(name, listener, view);

tracker.open();
```

Recovery

- Shared VM
- Java Security gives no protection against Out Of Memory
- CrashOnly
  - Dave Patterson, CS researcher at Berkely
  - Having Built A Resilient Infrastructure That Can Cope With Change And Accepts Dynamicity Then We Can Deal Effectively With The Concept Of Failure

Conclusions

- In order to build scalable applications you need to use the right patterns
- Patterns should be simple (i.e. expose relevant problems to relevant people)
- Model Driven Patterns Are Easier To Introspect And Change

Conclusions

- Expect failure at larger scales
- Even small problems will be magnified by scale

Conclusions

- Dynamic behaviours can increase resilience
- Resilience cannot be achieved via forcing the world to be the way you want it
- King Canute and his enterprise?

Summary

- Large scale enterprise applications can be built using OSGi
- and you can deploy them on Newton and Infiniflow ;)

Questions?

- Contact info:
  - dave.savage@paremus.com
  - http://newton.codecauldron.org
  - http://www.paremus.com
  - Exhibitor Stand 400 (located just by the doorway)
  - Come and see a demo.