Building dynamic distributed data stores with OSGi

Tim Ward

http://www.paremus.com
info@paremus.com
Who is Tim Ward?
@TimothyWard

• Senior Consulting Engineer and Architect at Paremus

• 5 years at IBM developing WebSphere Application Server
  • Container Implementation experience with Java EE and OSGi, including Blueprint, JPA, EJB and JTA

• OSGi Specification lead for JPA and Bytecode Weaving

• PMC member of the Apache Aries project

• Previous speaker at EclipseCon, Devoxx, Jazoon, JAX London, OSGi Community Event...

• Author of Manning’s Enterprise OSGi in Action
  • http://www.manning.com/cummins
What we’re going to cover

• OSGi Remote Services

• Deploying external processes using Packager

• Building Dynamically wired distributed systems

• Demo
OSGi Remote Services
OSGi Remote Services (1)

• OSGi Bundles communicate using the Service Registry
OSGi Remote Services (1)

- OSGi Bundles communicate using the Service Registry
- This allows for dynamic, loosely coupled interaction
OSGi Remote Services (1)

- OSGi Bundles communicate using the Service Registry
- This allows for dynamic, loosely coupled interaction
OSGi Remote Services (1)

- OSGi Bundles communicate using the Service Registry
- This allows for dynamic, loosely coupled interaction
Because OSGi services are loosely coupled they can come from anywhere!
OSGi Remote Services (2)

• Because OSGi services are loosely coupled they can come from anywhere!
• Any service can be transparently remoted *
OSGi Remote Services (2)

• Because OSGi services are loosely coupled they can come from anywhere!
  • Any service can be transparently remoted *
  • Service properties are made available remotely too
OSGi Remote Services (2)

- Because OSGi services are loosely coupled they can come from anywhere!
- Any service can be transparently remoted *
- Service properties are made available remotely too

Diagram:

```
Provider A

Network

Wormhole
```

*Remoting made transparent by the Wormhole service.
OSGi Remote Services (2)

• Because OSGi services are loosely coupled they can come from anywhere!
• Any service can be transparently remoted *
• Service properties are made available remotely too
OSGi Remote Services (2)

- Because OSGi services are loosely coupled they can come from anywhere!
- Any service can be transparently remoted *
- Service properties are made available remotely too

* Make sure your service is actually suitable for remoting!
OSGi Remote Services (3)

- Exposing a remote service is not transparent
Exposing a remote service is not transparent

```
objectclass = {List, Collection}
```

Diagram showing:
- Consumer
- Network
- Wormhole
- Provider A

Connected by arrows indicating the flow of information.
OSGi Remote Services (3)

- Exposing a remote service is not transparent
- Services opt-in using the `service.exported.interfaces` property

```
objectclass = {List, Collection}
```

```
Consumer
```

```
Network
```

```
Wormhole
```

```
Provider A
```

Wednesday, 30 October 13
OSGi Remote Services (3)

• Exposing a remote service is not transparent
• Services opt-in using the `service.exported.interfaces` property
OSGi Remote Services (3)

- Exposing a remote service is not transparent
- Services opt-in using the `service.exported.interfaces` property
- Imported services are marked with a `service.imported = true`

Consumer

```plaintext
objectclass = {Collection}
service.imported = true
```

Wormhole

```
Network

objectclass = {List, Collection}
```

Provider A

```
objectclass = {Collection}
service.exported.interfaces = {Collection}
```
OSGi Remote Services and Remote Service Admin
• The OSGi Alliance has two specifications. Remote Services and Remote Service Admin (RSA)
OSGi Remote Services and Remote Service Admin

• The OSGi Alliance has two specifications. Remote Services and Remote Service Admin (RSA)

• What I have described is Remote Services
The OSGi Alliance has two specifications. Remote Services and Remote Service Admin (RSA)

What I have described is Remote Services

The underlying implementation is completely unspecified
The OSGi Alliance has two specifications. Remote Services and Remote Service Admin (RSA)

What I have described is Remote Services

The underlying implementation is completely unspecified

RSA builds on Remote Services, and is more prescriptive
OSGi Remote Services and Remote Service Admin

• The OSGi Alliance has two specifications. Remote Services and Remote Service Admin (RSA)

• What I have described is Remote Services
  • The underlying implementation is completely unspecified

• RSA builds on Remote Services, and is more prescriptive
  • Pluggable distribution mechanisms (WS, RMI, JMS...)
The OSGi Alliance has two specifications. Remote Services and Remote Service Admin (RSA)

What I have described is Remote Services

The underlying implementation is completely unspecified

RSA builds on Remote Services, and is more prescriptive

Pluggable distribution mechanisms (WS, RMI, JMS...)

Pluggable discovery mechanisms (SLP, mDNS, Zookeeper...)

Wednesday, 30 October 13
OSGi Remote Services and Remote Service Admin

- The OSGi Alliance has two specifications. Remote Services and Remote Service Admin (RSA)
  - What I have described is Remote Services
    - The underlying implementation is completely unspecified
  - RSA builds on Remote Services, and is more prescriptive
    - Pluggable distribution mechanisms (WS, RMI, JMS...)
    - Pluggable discovery mechanisms (SLP, mDNS, Zookeeper...)
    - Notifications for “Topology Manager” bundles to decide which services to export/import
Packaging Applications with Packager
Packager Aims and advantages
Packager Aims and advantages

• Package programs as self-describing OSGi Bundles
Packager Aims and advantages

- Package programs as self-describing OSGi Bundles
- Allows Dynamic Resolution and Assembly of runtimes
Packager Aims and advantages

- Package programs as self-describing OSGi Bundles
- Allows Dynamic Resolution and Assembly of runtimes
- Can be semantically versioned
Packager Aims and advantages

• Package programs as self-describing OSGi Bundles
  • Allows Dynamic Resolution and Assembly of runtimes
  • Can be semantically versioned

• Leverage the existing bundle/service lifecycle and API
Packager Aims and advantages

- Package programs as self-describing OSGi Bundles
  - Allows Dynamic Resolution and Assembly of runtimes
  - Can be semantically versioned

- Leverage the existing bundle/service lifecycle and API
  - Allow processes to be installed and uninstalled in a running system
Packager Aims and advantages

- Package programs as self-describing OSGi Bundles
  - Allows Dynamic Resolution and Assembly of runtimes
  - Can be semantically versioned

- Leverage the existing bundle/service lifecycle and API
  - Allow processes to be installed and uninstalled in a running system
  - Allow processes to be dynamically started and stopped
Packager Aims and advantages

• Package programs as self-describing OSGi Bundles
  • Allows Dynamic Resolution and Assembly of runtimes
  • Can be semantically versioned

• Leverage the existing bundle/service lifecycle and API
  • Allow processes to be installed and uninstalled in a running system
  • Allow processes to be dynamically started and stopped

• Allow integration with other OSGi specification
Packager Aims and advantages

• Package programs as self-describing OSGi Bundles
  • Allows Dynamic Resolution and Assembly of runtimes
  • Can be semantically versioned

• Leverage the existing bundle/service lifecycle and API
  • Allow processes to be installed and uninstalled in a running system
  • Allow processes to be dynamically started and stopped

• Allow integration with other OSGi specification
  • Dynamic Configuration using Config Admin,
Packager Aims and advantages

• Package programs as self-describing OSGi Bundles
  • Allows Dynamic Resolution and Assembly of runtimes
  • Can be semantically versioned

• Leverage the existing bundle/service lifecycle and API
  • Allow processes to be installed and uninstalled in a running system
  • Allow processes to be dynamically started and stopped

• Allow integration with other OSGi specification
  • Dynamic Configuration using Config Admin,
  • Use and provide Local and Remote Services
Packager Aims and advantages

• Package programs as self-describing OSGi Bundles
  • Allows Dynamic Resolution and Assembly of runtimes
  • Can be semantically versioned

• Leverage the existing bundle/service lifecycle and API
  • Allow processes to be installed and uninstalled in a running system
  • Allow processes to be dynamically started and stopped

• Allow integration with other OSGi specification
  • Dynamic Configuration using Config Admin,
  • Use and provide Local and Remote Services

• Allow existing code to be packaged without change
Approach
Approach

• Wrap Native Artifacts in OSGi Bundles
Approach

- Wrap Native Artifacts in OSGi Bundles
- Link the Artifact Lifecycle to OSGi.
Approach

• Wrap Native Artifacts in OSGi Bundles

• Link the Artifact Lifecycle to OSGi.
  • Bundle Install/Resolve/Start => Artifact “install”
**Approach**

- Wrap Native Artifacts in OSGi Bundles
- **Link the Artifact Lifecycle to OSGi.**
  - Bundle Install/Resolve/Start $\Rightarrow$ Artifact “install”
  - Service registration/unregistration $\Rightarrow$ Artifact “run/stop”
Approach

• Wrap Native Artifacts in OSGi Bundles

• Link the Artifact Lifecycle to OSGi.
  • Bundle Install/Resolve/Start => Artifact “install”
  • Service registration/unregistration => Artifact “run/stop”
  • Bundle Uninstall => Artifact “uninstall”
Approach

• Wrap Native Artifacts in OSGi Bundles

• **Link the Artifact Lifecycle to OSGi.**
  - Bundle Install/Resolve/Start => Artifact “install”
  - Service registration/unregistration => Artifact “run/stop”
  - Bundle Uninstall => Artifact “uninstall”

• Link to Standard OSGi Services: Configuration Admin, Metatype, Log Service...
Approach

• Wrap Native Artifacts in OSGi Bundles

• Link the Artifact Lifecycle to OSGi.
  • Bundle Install/Resolve/Start => Artifact “install”
  • Service registration/unregistration => Artifact “run/stop”
  • Bundle Uninstall => Artifact “uninstall”

• Link to Standard OSGi Services: Configuration Admin, Metatype, Log Service...

```xml
<system name="BackEnd:MongoDB" boundary="fibre">
  <!-- MongoDB package -->
  <system.part category="msf" name="com.paremus.packager.demos.mongo.guard">
    <property name="port" value="27017" />
  </system.part>
</system>
```
Separation of Concerns
Separation of Concerns

• What are we running, and how do we configure/run it?
Separation of Concerns

• What are we running, and how do we configure/run it?
• Package Type Service
Separation of Concerns

- What are we running, and how do we configure/run it?
- **Package Type Service**

- When should we start it, and what configuration properties should we use?
Separation of Concerns

- What are we running, and how do we configure/run it?
  - Package Type Service

- When should we start it, and what configuration properties should we use?
  - Process Guard Service
Separation of Concerns

• What are we running, and how do we configure/run it?
  • Package Type Service

• When should we start it, and what configuration properties should we use?
  • Process Guard Service

• Actually invoke the start/stop scripts, and monitor the state of the external process
Separation of Concerns

• What are we running, and how do we configure/run it?
  • Package Type Service

• When should we start it, and what configuration properties should we use?
  • Process Guard Service

• Actually invoke the start/stop scripts, and monitor the state of the external process
  • Packager Manager/Watchdog Process
Packager Package Types
Package Type

• Usually **contains** the native parts
Package Type

- Usually **contains** the native parts
- **Installs** the Native Program
Package Type

• Usually **contains** the native parts
• **Installs** the Native Program
• Configures the program based on the supplied config
Package Type

- Usually **contains** the native parts
- **Installs** the Native Program
- Configures the program based on the supplied config
- Returns **scripts** to the Packager for: start, stop, ping...
Package Type

- Usually **contains** the native parts
- **Installs** the Native Program
- Configures the program based on the supplied config
- Returns **scripts** to the Packager for: start, stop, ping...
- Is (usually) **platform specific**
Package Type

- Usually **contains** the native parts
- **Installs** the Native Program
- Configures the program based on the supplied config
- Returns **scripts** to the Packager for: start, stop, ping...
- Is (usually) **platform specific**

**Provide-Capability:** packager.type;
    packager.type=mongodb;
    version:Version=2.2.0

**Require-Capability:** osgi.native;
    filter:="(&
        (osgi.native.osname=Linux)
        (osgi.native.processor=x86-64)
    )"
How the Package Type works
How the Package Type works

• Installing an external process can take a long time
How the Package Type works

• Installing an external process can take a long time
• Not suitable for running in a bundle’s Activator!
How the Package Type works

• Installing an external process can take a long time
  • Not suitable for running in a bundle’s Activator!
• Installation should be asynchronous with respect to bundle start
How the Package Type works

• Installing an external process can take a long time
  • Not suitable for running in a bundle’s Activator!
• Installation should be asynchronous with respect to bundle start
  • Run on a separate thread?
How the Package Type works

• Installing an external process can take a long time
  • Not suitable for running in a bundle’s Activator!
• Installation should be asynchronous with respect to bundle start
  • Run on a separate thread?
  • Install on demand?
How the Package Type works

• Installing an external process can take a long time
  • Not suitable for running in a bundle’s Activator!
• Installation should be asynchronous with respect to bundle start
  • Run on a separate thread?
  • Install on demand?
• Notify possible clients when a type is “ready” using the whiteboard pattern
How the Package Type works

• Installing an external process can take a long time
  • Not suitable for running in a bundle’s Activator!
• Installation should be asynchronous with respect to bundle start
  • Run on a separate thread?
  • Install on demand?
• Notify possible clients when a type is “ready” using the whiteboard pattern
  • Service properties advertise what the external process is
How the Package Type works

• Installing an external process can take a long time
  • Not suitable for running in a bundle’s Activator!
• Installation should be asynchronous with respect to bundle start
  • Run on a separate thread?
  • Install on demand?
• Notify possible clients when a type is “ready” using the whiteboard pattern
  • Service properties advertise what the external process is
  • Service unregisters to “uninstall”
Packager Process Guards
Process Guard Service

- **Gathers** Process Config Properties
Process Guard Service

• **Gathers** Process Config Properties

• Existence signals **When** to Start/Stop
Process Guard Service

- **Gathers** Process Config Properties
- Existence signals **When** to Start/Stop
- Receives State Change **Events** (started, crashed...)

Wednesday, 30 October 13
Process Guard Service

- **Gathers** Process Config Properties
- Existence signals **When** to Start/Stop
- Receives State Change **Events** (started, crashed...)
- **Advertises** the Running Package (Optional)
Process Guard Service

- **Gathers** Process Config Properties
- Existence signals **When** to Start/Stop
- Receives State Change **Events** (started, crashed...)
- **Advertises** the Running Package (Optional)
- Usually platform independent
How the Process Guard works
How the Process Guard works

- Most external processes require some level of configuration to run
How the Process Guard works

• Most external processes require some level of configuration to run
• This configuration is often platform independent
How the Process Guard works

• Most external processes require some level of configuration to run
  • This configuration is often platform independent
• When a process should be started the process guard registers a service
How the Process Guard works

• Most external processes require some level of configuration to run
  • This configuration is often platform independent
• When a process should be started the process guard registers a service
  • Another example of using the whiteboard pattern
How the Process Guard works

• Most external processes require some level of configuration to run
  • This configuration is often platform independent
• When a process should be started the process guard registers a service
  • Another example of using the whiteboard pattern
• It uses the same service properties as the package type
How the Process Guard works

• Most external processes require some level of configuration to run
  • This configuration is often platform independent
• When a process should be started the process guard registers a service
  • Another example of using the whiteboard pattern
• It uses the same service properties as the package type
• Often implemented as a ManagedService or ManagedServiceFactory
How the Process Guard works

• Most external processes require some level of configuration to run
  • This configuration is often platform independent
• When a process should be started the process guard registers a service
  • Another example of using the whiteboard pattern
  • It uses the same service properties as the package type
• Often implemented as a ManagedService or ManagedServiceFactory
  • Automatically Inject/Delete the configuration for the process
How the Process Guard works

• Most external processes require some level of configuration to run
  • This configuration is often platform independent
• When a process should be started the process guard registers a service
  • Another example of using the whiteboard pattern
• It uses the same service properties as the package type
• Often implemented as a ManagedService or ManagedServiceFactory
  • Automatically Inject/Delete the configuration for the process
How the Process Guard works

• Most external processes require some level of configuration to run
  • This configuration is often platform independent
• When a process should be started the process guard registers a service
  • Another example of using the whiteboard pattern
• It uses the same service properties as the package type
• Often implemented as a ManagedService or ManagedServiceFactory
• Automatically Inject/Delete the configuration for the process
The Packager Manager
The Packager Manager

- Listens for matching Package Type and Process Guard Services
The Packager Manager

• Listens for matching Package Type and Process Guard Services
• One Package Type may match many Process Guards
The Packager Manager

- Listens for matching Package Type and Process Guard Services
- One Package Type may match many Process Guards
- For every pair the packager manager starts an instance of the process
The Packager Manager

• Listens for matching Package Type and Process Guard Services
  • One Package Type may match many Process Guards

• For every pair the packager manager starts an instance of the process
  • Configured by the Package Type with properties from the Process Guard
The Packager Manager

- Listens for matching Package Type and Process Guard Services
- One Package Type may match many Process Guards

- For every pair the packager manager starts an instance of the process
  - Configured by the Package Type with properties from the Process Guard

- A ‘Watchdog’ monitors the process (using scripts from the package type)
The Packager Manager

• Listens for matching Package Type and Process Guard Services
  • One Package Type may match many Process Guards

• For every pair the packager manager starts an instance of the process
  • Configured by the Package Type with properties from the Process Guard

• A ‘Watchdog’ monitors the process (using scripts from the package type)
  • Sends notifications to the Process Guard about the state of the process
The Packager Manager

• Listens for matching Package Type and Process Guard Services
  • One Package Type may match many Process Guards

• For every pair the packager manager starts an instance of the process
  • Configured by the Package Type with properties from the Process Guard

• A ‘Watchdog’ monitors the process (using scripts from the package type)
  • Sends notifications to the Process Guard about the state of the process

• If either service is unregistered then the process is shut down
Packager Interactions

Packager Manager

Package Type

Register services

Process Guard
Packager Interactions

Packager Manager

Package Type

Get Configuration

Process Guard
Packager Interactions

Packager Manager

Package Type

Create Scripts and start!

Process Guard
Packager Interactions

Packager Manager

Package Type

Monitor and notify

Process Guard

Wednesday, 30 October 13
Packager Interactions

Packager Manager

Package Type

Shutdown on unregistration

Process Guard
Building dynamically wired systems with Remote Services and Packager
Publishing an Endpoint

• How do you tell the rest of the framework that your external process has started, or where it is?
Publishing an Endpoint

• How do you tell the rest of the framework that your external process has started, or where it is?

• The whiteboard pattern to the rescue again!
Publishing an Endpoint

• How do you tell the rest of the framework that your external process has started, or where it is?

• The whiteboard pattern to the rescue again!

• The Process Guard can register an “Endpoint” with a marker interface and a service property with a URI to the process
Publishing an Endpoint

• How do you tell the rest of the framework that your external process has started, or where it is?

• The whiteboard pattern to the rescue again!

• The Process Guard can register an “Endpoint” with a marker interface and a service property with a URI to the process

• It can also make this remotely available...
Publishing an Endpoint

• How do you tell the rest of the framework that your external process has started, or where it is?
  • The whiteboard pattern to the rescue again!

• The Process Guard can register an “Endpoint” with a marker interface and a service property with a URI to the process
  • It can also make this remotely available...

• Suddenly the process is discoverable in every framework
Dynamically building configurations
Dynamically building configurations

• Process Guard services can be very simple
Dynamically building configurations

- Process Guard services can be very simple
- Static properties
Dynamically building configurations

• Process Guard services can be very simple
• Static properties
• Injection from Config Admin
Dynamically building configurations

- Process Guard services can be very simple
  - Static properties
  - Injection from Config Admin

- They can also wait and listen for Endpoints (local and remote!)
Dynamically building configurations

• Process Guard services can be very simple
  • Static properties
  • Injection from Config Admin

• They can also wait and listen for Endpoints (local and remote!)
  • Trigger a ManagedServiceFactory with part of the configuration
Dynamically building configurations

• Process Guard services can be very simple
  • Static properties
  • Injection from Config Admin

• They can also wait and listen for Endpoints (local and remote!)
  • Trigger a ManagedServiceFactory with part of the configuration
    • Listen for the Endpoint, complete the configuration
Dynamically building configurations

- Process Guard services can be very simple
  - Static properties
  - Injection from Config Admin

- They can also wait and listen for Endpoints (local and remote!)
  - Trigger a ManagedServiceFactory with part of the configuration
    - Listen for the Endpoint, complete the configuration
    - Register the Process Guard!
Dynamically building configurations (2)
Dynamically building configurations (2)

• Replicating this out over multiple nodes lets you dynamically assemble large systems with little to no manual configuration input.
Dynamically building configurations (2)

• Replicating this out over multiple nodes lets you dynamically assemble large systems with little to no manual configuration input

• Java EE servers
Dynamically building configurations (2)

• Replicating this out over multiple nodes lets you dynamically assemble large systems with little to no manual configuration input

• Java EE servers

• Messaging infrastructure
Dynamically building configurations (2)

• Replicating this out over multiple nodes lets you dynamically assemble large systems with little to no manual configuration input

• Java EE servers

• Messaging infrastructure

• NoSQL stores
Dynamically building configurations (2)

• Replicating this out over multiple nodes lets you dynamically assemble large systems with little to no manual configuration input

• Java EE servers

• Messaging infrastructure

• NoSQL stores

• Because OSGi services are dynamic we can track the service to see if the endpoint moves!
Dynamically building configurations (2)

• Replicating this out over multiple nodes lets you dynamically assemble large systems with little to no manual configuration input

• Java EE servers

• Messaging infrastructure

• NoSQL stores

• Because OSGi services are dynamic we can track the service to see if the endpoint moves!

•Automatic reconfiguration can rewire the system if a node fails!
Putting Packager to the test with Redis
What is Redis?
What is Redis?

- Redis is an open source key-value store (http://redis.io)
What is Redis?

- Redis is an open source key-value store (http://redis.io)

- Redis supports queries based on key or value data
What is Redis?

- Redis is an open source key-value store ([http://redis.io](http://redis.io))

- Redis supports queries based on key or value data
  - Values can be rich objects like hashes, lists sets or sorted sets
What is Redis?

- Redis is an open source key-value store (http://redis.io)
- Redis supports queries based on key or value data
  - Values can be rich objects like hashes, lists sets or sorted sets
- Redis supports replication to read-only “slave” nodes
What is Redis?

- Redis is an open source key-value store (http://redis.io)

- Redis supports queries based on key or value data
  - Values can be rich objects like hashes, lists sets or sorted sets

- Redis supports replication to read-only “slave” nodes
  - Wouldn’t it be nice to automatically configure the slaves?
Demo
Thanks!

- For more about OSGi...
  - Specifications at [http://www.osgi.org](http://www.osgi.org)
  - Enterprise OSGi in Action
    - [http://www.manning.com/cummins](http://www.manning.com/cummins)

- For more about Packager...
  - [http://www.bundlerrepo.org:8090/display/SF110/Packager](http://www.bundlerrepo.org:8090/display/SF110/Packager)

Questions?

[http://www.paremus.com](http://www.paremus.com)
info@paremus.com