Is OSGi Modularity Always Worth It?

Glyn Normington
Agenda

• Costs and benefits
• Case studies
• When is OSGi worth it?
OSGi Benefits

✓ Encapsulated module internals
✓ Easier to understand, maintain, and extend
✓ Explicit, versioned dependencies
✓ Simplified impact analysis
✓ Faster class loading
✓ Visible architecture
✓ bundles, services, dependencies
✓ execution environment
OSGi Benefits

✓ Side-by-side versioning
✓ Dynamism
✓ Loose coupling via services
OSGi Costs

- **Learning: OSGi, new runtime, new tools**
- **Reusing 3rd party JARs**
  - Designing appropriate bundles and services
  - Defining and upgrading dependencies
  - Building bundles
  - Debugging resolution failures and class loading
  - Coping with dynamism
vert.x Case Study
vert.x

- Asynchronous, event-driven web apps
- HTTP, WebSockets, sockjs
- Apps in Java, Groovy, JavaScript, Ruby
- **Apps need not be thread safe**
- JSON event bus
Vert.x diagram

- **Client**: Connects to the server via HTTP, WebSockets, and sockjs

- **Server**:Centralizes communication via the event bus

- **Verticle**: Executed by the server for processing

- **Module**: Part of the verticle, possibly containing logic

- **Busmod**: Another module, possibly for event handling

- **Class Loader**: Loads classes for verticle and busmod execution

- **Event Bus**: Facilitates communication between verticles and modules
Converting vert.x to OSGi

- Convert vert.x runtime to bundles
- Convert verticles to bundles
- (Convert busmods to bundles - similarly)
Convert Runtime to Bundles

- JARs:
  - vert.x core
  - Netty - network IO
  - Jackson & Jackson Mapper - JSON
Approach

• Eye-ball existing OSGi manifests
• Generate manifests where necessary
  • Use bnd or Virgo bundlor
• Create a manifest template
• Test resolution
Existing OSGi Manifests

✓ Netty
  - Bundle version 3.4.2.Final

✓ Jackson & Jackson Mapper
  - Bundle version 1.9.4
Bundle vert.x core

- Use bnd or Virgo bundlor
- Manifest template required ...
vert.x Core Template

- vert.x name and version

Bundle-ManifestVersion: 2

Bundle-SymbolicName: org.vertx.core

Bundle-Name: vert.x Core

Bundle-Version: 1.0.0.final

Export-Template: *;version="1.0.0.final"
vert.x Core Template

- Package import template version ranges
  - Netty: [3.4.2.Final, 4.0)
  - Jackson: [1.9.4, 2.0)
- Other optional imports: 0
  - 0 means [0, ∞)
vert.x Core Template

• Package import version ranges
  • Netty: [3.4.2.Final, 4.0)
  • Jackson: [1.9.4, 2.0)
  • Other optional imports: 0
    • 0 means [0, ∞)

Where did these come from?
Version Ranges

• Lower bound
  • What did you test?
  • What old versions do you support?

• Upper bound
  • Exporter’s versioning policy?
  • How easy is it to change your bundle?
    • Lifespan?
  • Distribution?
Netty Version Range

@glynnormington 3.x is compatible in the complete series.. 4.x is not compatible with 3.x
04:08 PM - 09 Oct 12

@glynnormington No strict rule but we should tell you when we announce a new version if binary compatibility is broken across minor updates
02:02 PM - 09 Oct 12

[3.4.2.Final, 4.0)  
[3.4.2.Final, 3.5)
Generate Bundle

`bundlor.sh -i vert.x-core.jar`
`-m vertx.core.mf`
`-o vert.x-core-1.0.0.final.jar`
Test Metadata

• Using Virgo, for simplicity
• Place dependencies in repository/usr
• Place vert.x core in pickup
• Start Virgo:

...<DE0005I> Started bundle 'org.vertx.core' version '1.0.0.final'.
Modular Verticles
public class MyVerticle extends Verticle {

    public void start() {

        vertx.createHttpServer().requestHandler(
            new Handler<HttpServerRequest>() {
                public void handle(HttpServerRequest req) {
                    ...
                }
            }
        ).listen(8080);
    }
}
public class MyVerticle extends Verticle {

    public void start() {
        vertx.createHttpServer().requestHandler(
            new Handler<HttpServerRequest>() {
                public void handle(HttpServerRequest req) {
                    ...
                }
            }
        ).listen(8080);
    }
}
public class MyVerticle extends Verticle {

    public void start() {
        vertx.createHttpServer().requestHandler(
            new Handler<HttpServerRequest>() {
                public void handle(HttpServerRequest req) {
                    ...
                }
            }).listen(8080);
    }
}

public class MyVerticle extends Verticle {

    public void start() {
        vertx.createHttpServer().requestHandler(
            new Handler<HttpServerRequest>() {
                public void handle(HttpServerRequest req) {
                    ...
                }
            }
        ).listen(8080);
    }
}

redundant boilerplate
not easily unit tested
public class MyVerticle extends Verticle {

    public void start() {

        vertx.createHttpServer().requestHandler(
            new Handler<HttpServerRequest>() {
                public void handle(HttpServerRequest req) {
                    ...
                }
            }
        ).listen(8080);
    }
}
Modular Verticle

• Move boilerplate code to a support bundle
• Convert configuration to declarative metadata
• Use services for loose coupling
  • whiteboard pattern
public class MyHandler implements Handler<HttpServerRequest> {

    public void handle(HttpServerRequest req) {
        ...
    }
}

<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
  <bean class="org.vertx.osgi.sample.MyHandler"
    id="handler"/>

  <service interface="org.vertx.java.core.Handler"
    ref="handler">
    <service-properties>
      <entry key="type" value="HttpServerRequestHandler"/>
      <entry key="port" value="8080"/>
    </service-properties>
  </service>
</blueprint>
1. listen for/find bundle
2. publish handler
3. listen for/find handler
4. create server
5. register handler
6. set server listening
7. request from network
8. Handler.handle()
9. HttpServerRequest operations

OSGi Service Registry
Was it worth it?
Benefits

✓ vertx implementation can be hidden
✓ Boilerplate factored out
✓ Unit testing simplified
✓ Configuration externalised
Costs

- Deciding dependency version ranges
- Reworking vert.x to improve encapsulation
- Programming model changed
- Shared dependencies must be thread safe
Costs

- Deciding dependency version ranges
- Reworking vert.x to improve encapsulation
- Programming model changed
- Shared dependencies must be thread safe

Conclusion: probably not worth it
vSphere Web Client Case Study
vSphere Web Client

- OSGi based web server
  - Virgo Server for Apache Tomcat
- Plugins allow users to extend the Web Client
  - Custom data
  - Custom relationships (for new/existing data)
Was it worth it?
Benefits

✓ Modular
✓ Extensible by users
✓ Internals not accessible to users
✓ Dynamically updateable
Costs

- Steep initial learning curve
- Investment required in Virgo tooling
- OSGi exposed to users
Costs

- Steep initial learning curve
- Investment required in Virgo tooling
- OSGi exposed to users

Conclusion: worth it
Croatian Telecom Case Study
Overview

- Virgo 3.0.3, EclipseLink, Atomikos, ActiveMQ, Tomcat clustering, Hazelcast
- In production for more than a year
- Last 7 months as a cluster
- Nearly 1 million user accounts
- 100s of web requests per second, peak
- Only one major outage, due to a Linux bug
Cited Benefits

✓ Better understand components you use
✓ Cleaner, higher quality code
✓ Better system architecture
✓ Reusable modules
Cited Costs

- Time to learn the intricacies of OSGi
- Time to integrate non-OSGi ready components
Cited Costs

- Time to learn the intricacies of OSGi
- Time to integrate non-OSGi ready components

Conclusion: worth it
Wrap-up
OSGi Costs

- **Learning: OSGi, new runtime, new tools**
- **Reusing 3rd party JARs**
- Designing appropriate bundles and services
- Defining and upgrading dependencies
- Building bundles
- Debugging resolution failures and class loading
- Coping with dynamism
- **Reduced isolation in some cases**
- **Further investment in tooling required**
**OSGi Benefits**

- Encapsulated module internals
- Explicit, versioned dependencies
- Side-by-side versioning
- Dynamism
- Visible architecture
- Loose coupling via services
- Better understanding of components used
- Reusable modules
When is OSGi worth it?

✓ Lower layers benefit most
✓ Control dependencies
✓ Hide internals
   Lower layers push higher layers to OSGi
?
Exposing OSGi to users
✓ Long-lived or critical applications
PRO TANTO QUID RETRIBUAMUS

For so much, what shall we give in return?

(wording from the edge of the Belfast £1 coin)
Further Information

• “Java Application Architecture” by Kirk Knoernschild

• http://www.osgi.org

• http://underlap.blogspot.co.uk/2012/06/osgi-case-study-modular-vertx.html

• http://vertx.io/

• http://underlap.blogspot.co.uk/2012/10/virgo-in-vsphere.html

Credits

• “Coins” by Rob Redwood

• “Benefits of using OSGi”
  [http://www.osgi.org/Technology/WhyOSGi](http://www.osgi.org/Technology/WhyOSGi)

• “One Pound” by Leo Reynolds