Virgo by Example

Florian Waibel, Markus Knauer
Survey Who has used…

... Virgo ?
... WebSockets ?
... Docker ?
... Gradle ?
... git ?
Who we are

Florian

Markus
Eclipse Virgo by Example: Game of Life

This is a proof of concept build of Conway's game of life made using websockets.

The server calculates and generates the next board, and the client draws whatever the server tells it to.
Roadmap

1. Setup Workspace - Intro to Virgo Tooling
2. Embed JavaScript based “Game-of-Life” Jenova
3. Investigate game engine lifecycle
4. Add custom OSGi Command
5. Communicate via OSGi EventAdmin
6. Configure WebSocket
7. Build and Run with Docker
Installing Tutorial Prerequisites

- eclipse
- VIRGO
- git
Prerequisite 1: The IDE

1. Eclipse IDE for Java EE Developers
   a. Virgo Tooling
   b. Docker Tooling

pre-packaged versions available!
Download prepackaged Eclipse

Go to http://gol.eclipsesource.com/downloads/
and download prepackaged Eclipse archive depending on OS

USB stick:

cp eclipse-jee-neon-M5-virgo-tutorial-macosx-cocoa-x86_64.tar.gz ~/
Install Eclipse

Eclipse Neon M5 with

- Docker Tooling

USB stick:

```shell
unzip eclipse-jee-neon-M5-virgo-tutorial-win32-x86_64.zip
tar zvxf eclipse-jee-neon-M5-virgo-tutorial-macosx-cocoa-x86_64.tar.gz
```
Prerequisite 2: Custom Virgo Runtime


USB stick:
```
cp virgo-gol-runtime.tar.gz ~/
```
Install Virgo Runtime

Eclipse Virgo 3.7.0.M02 with

- Spring 4.2.1.RELEASE
- Nashorn JavaScript engine
- JSON Mapper Jackson ([https://github.com/FasterXML/jackson](https://github.com/FasterXML/jackson))

**USB stick:**
unzip virgo-gol-runtime.zip
tar xvfz virgo-gol-runtime.tar.gz
Verify Virgo Runtime Setup

- **Grant access to OSGi console** `${VIRGO_HOME}/repository/ext/osgi.console.properties`
- **Start Virgo Runtime**
  
  ${VIRGO_HOME}/bin/startup.sh

- **Go to Virgo Admin Console**
  
  http://localhost:8080/admin/ (admin/admin)

- **Connect to User Region via Telnet / SSH**
  
  telnet localhost 2501

  ssh -p 2502 admin@localhost (pw: admin)
Prerequisite 3: The Git Repo

git clone https://github.com/eclipsesource/virgo_game_of_life.git

USB Stick (Get local copy of the Git repository)

unzip virgo_game_of_life.zip -d ~/git/
10,000 Feet: Data Flow

Game of Life Backend

WebSocket

send click events

push updates

Browsers
5,000 Feet: Docker Deployment

Server

Docker

:B8080

:B8080

Browsers
3,000 Feet: Backend

Virgo Runtime

OSGi Bundles

OSGi Shell

Event Bus

OSGi Web Application Bundles

Servlet Engine
1,000 Feet: OSGi

- game API
- jenova
- OSGi commands
- game engine
- OSGi Event Admin
- static resources
  - /static
- server
  - /gol
Ready, Steady, Vir...  

Go!
Task 1: Workspace + API Bundle

game API

OSGi commands

game engine

jenova

OSGi Event Admin

static resources

/server

/gol
Tutorial as Branches

During the Tutorial YOU do:

1. Try to solve the tasks  
   (Hint: Look for TODO task_x.y in the code)
2. git diff task_x_<task_name>_final
3. git checkout task_x+1_<task_name>_begin
Start Game-of-Life Workspace

git checkout task_01_workspace_begin

USB Stick (Get local copy of Gradle dependencies)
unzip gradle-cache.
zip -d ~/.gradle/

Save your version of the cache

Create Eclipse Project Metadata
$ ./gradlew eclipse

Import... Gradle Project...
Import OSGi Bundle Projects

1. Switch to initial branch in your Git repo
   $ cd virgo_game_of_life
   $ git checkout task_01_workspace_begin

2. Create Eclipse Project Metadata
   $ ./gradlew eclipse

3. Start provided Eclipse with new workspace

4. Import... Gradle Project...
Create New Server Runtime

1. Open the Servers View
2. Select Virgo Runtime
3. Select path to Virgo Runtime
Configure Server Runtime

Drop your bundles onto server

Double-click on server, adjust publishing settings
task 01.1 Fix template.mf

**General Information**
This section describes general information about this bundle

- **ID:** `com.eclipsesource.examples.gol.api`
- **Version:** 0.1
- **Name:** TODO task 01.1 provide bundle name and check in OSGi console with headers command

**Vendor:**

**Activator:**
Verify Game-of-Life Workspace

```
.osgi> ss api
"Framework is launched."

id     State     Bundle
126     ACTIVE    com.eclipsesource.examples.gol.api_0.1.0

.osgi> headers 126
Bundle headers:
...
  Bundle-Name = Game of Life API
...
```
Task 2: Jenova - JavaScript

- OSGi commands
- game API
- game engine
- Jenova - JavaScript game API
- static resources
  - /static
- server
  - /gol
git checkout task_02_jenova_begin

Create Eclipse Project Metadata

$ ./gradlew eclipse

Import... Gradle Project...
JSR-223 based mechanism for scripted beans, exposed through the `<lang:std />` element in XML. (backed by the `StandardScriptFactory`)
Expose a referenced Spring bean as OSGi service with a given interface with the `<osgi:service>` element in XML.
Task 2: Embedded JavaScript

02.1 add id 'jenova' and specify the matching Java interface
02.2 merge jenova.js here and verify result with JUnit test JenovaTest
02.3 expose JavaScript backed Jenova bean as OSGi service and verify result via OSGi console
Verify Green JUnit tests + Console

$ ./gradlew :jenova:test

osgi> services *GameOfLife

{com.eclipsesource.examples.gol.api.GameOfLife}={org.eclipse.gemini.blueprint.bean.
name=jenova, ..., Bundle-SymbolicName=com.eclipsesource.examples.gol.jenova, Bundle-
Version=0.1.0, service.id=251}

"Registered by bundle:" com.eclipsesource.examples.gol.jenova_0.1.0 [127]
End Jenova - Embedded JavaScript

git diff task_02_jenova_final
Bonus Jenova - JavaScript

? Consume the JavaScript snippet from the file system (i.e. not inlined in the XML)
Task 3: NanoService GameEngine

game API

jenova

game engine

OSGi commands

OSGi Event Admin

static resources
/static

server
/gol
Start NanoService GameEngine

git checkout task_03_engine_begin

Create Eclipse Project Metadata

$ ./gradlew eclipse

Import... Gradle Project...
Publishes an OSGi reference as Spring bean named jenova with a given interface and the `<osgi:reference />` element in XML.
package c.e.e.gol.engine;

@Component("gameEngine")
public class DefaultGameEngine {
    @Autowired
    private GameOfLife gameOfLife;

    @PostConstruct
    public void init() {};

    @PreDestroy
    public void destroy() {};
}

<context:component-scan
    base-package="c.e.e.gol.engine" />

- Name of the Spring component
- Inject GameOfLife bean
- Spring bean lifecycle hooks
- All classes within the base package will be processed by Spring
Task 3: Nano service GameEngine

03.1 autowire GameOfLife
03.2 start bean post construction
03.3 calculate and store next generation of the board
03.4 shutdown bean pre destruction
03.5 enable component scan for bundle game engine
03.6 reference OSGi service GameOfLife as bean with id jenova
03.7 publish GameEngine as OSGi service
Verify NanoService GameEngine

$ tail -f ${VIRGO_HOME}/serviceability/logs/log.log
-- Calculating next generation --
-- Calculating next generation --
-- Calculating next generation --
...

osgi> services *GameEngine
?

End NanoService GameEngine

git diff task_03_engine_final
Bonus NanoService GameEngine

Solve the “task” without Annotations - only XML
Task 4: OSGi Game Commands

game API

jenova

game engine

OSGi commands

OSGi Event Admin

static resources

/server

/static
Start Custom OSGi Commands

git checkout task_04_commands_begin

Create Eclipse Project Metadata

$ ./gradlew eclipse

Import... Gradle Project...
Custom OSGi Commands

Provide “add” as OSGi commands

```java
public class OsgiCommandProvider implements CommandProvider {

    public Object _add(CommandInterpreter commandInterpreter) {
        gameEngine.addObject(...);
        return null;
    }

    public String getHelp() {
        return "...";
    }
}
```

All methods starting with an underscore like “_add” will be available as OSGi commands.
Task 4: OSGi Game Commands

04.1 reference OSGi service `GameEngine`

04.2 implement OSGi command:

```
add [object_name] [x[,y]]
```

Hint: Predefined patterns are in `OsgiCommandProvider`
Verify Talk to your App on the Shell

```
osgi> init 10 5
```
```
osgi> print
  01000000001
  0100001000
  0000001011
  0000001001
  0100001111
```
```
osgi> reset
```
```
osgi> add blinker 5 1
```
```
osgi> print
  0000000000
  0000010000
  0000010000
  0000010000
  0000000000
```
End Custom OSGi Commands

git diff task_04_commands_final
Bonus Custom OSGi Commands

Add LWSS (Light Weight Space Ship)
Add command to flip the board vertically
Task 5: Publish Events
Start OSGi Event Admin

git checkout task_05_events_begin

Create Eclipse Project Metadata

$ ./gradlew eclipse

Import... Gradle Project...
Expose a referenced Spring bean as OSGi EventHandler listening for topic "topic_foo" with the <service-properties /> element in XML.
Task 5: Publish / Subscribe Events

05.1 `autowire EventAdmin`

05.2 post event "topic_newBoard" with key="board" and payload board

05.3 post event "topic_userModifiedCell" and keys "x", "y"

05.4 register bean moveListener as OSGi service EventHandler for "topic_newBoard" events...

05.5 … and "topic_userModifiedCell" events

05.6 Print events to `System.out` in `MoveListenerDelegate.handleEvent()`
$ tail -f ${VIRGO_HOME}/serviceability/logs/log.log

-- Calculating next generation --
Event arrived: o.o.s.e.Event [topic=topic_newBoard]
Available properties: [board, event.topics]

-- Calculating next generation --
Event arrived: o.o.s.e.Event [topic=topic_newBoard]
Available properties: [board, event.topics]
...

Verify EventAdmin
Task 6: WebSocket

Don't forget to provide these OSGi services:
Start WebSocket

git checkout task_06_websocket_begin

Create Eclipse Project Metadata

$ ./gradlew eclipse

Import... Gradle Project...
<websocket:message-broker application-destination-prefix="/app">
    <websocket:stomp-endpoint path="/ws">
        <websocket:sockjs />
    </websocket:stomp-endpoint>
    <websocket:simple-broker prefix="/topic" />
</websocket:message-broker>

Creates bean SimpMessagingTemplate

Stomp - text orientated messaging protocol ([http://stomp.github.io/](http://stomp.github.io/))
SockJS - mimics the WebSockets API, but instead of WebSocket there is a SockJS Javascript object. ([http://sockjs.org](http://sockjs.org))
@Controller("app")
public class App {

    @MessageMapping("/updateCell")
    public void updateCell(Cell cell) {
        // handle incoming message
    }

    @RequestMapping(value = "/board", method = RequestMethod.GET)
    public String board() {
        return "board";
    }
}

Client to Server

Called when a user “toggles” a cells

Initial request from the browser
Server to Client

```java
@Component("topic")
public class Topic {
    @Autowired
    private SimpMessagingTemplate template;

    public void next(int[][] board) {
        template.convertAndSend("/topic/newBoard", board);
    }
}
```
Task 6: Websockets

06.1 add message mapping for "/updateCell"
06.2 post event "topic_updateCell" with "x" and "y" coordinates; server side event handling missing
06.3 auto wire SimpMessagingTemplate
06.4 convert and send board to "topic/newBoard"
06.5 convert and send cell to "topic/userModifiedCell"
06.6 register stomp endpoint with SockJS support
06.7 replace NOP implementation in MoveListenerDelegate. handleEvent()
06.8 implement EventHandler for toggling in DefaultGameEngine to enable client to server communication
Verify WebSocket

Browse to http://localhost:8080/gol/board
git diff task_06_websocket_final
Task 7: Docker Deployment

Server

Docker

:B8080
:B8080

Browsers
Start Deployment

git checkout task_07_docker_begin

USB Stick (Get local copy of base image)

$ cat java_openjdk-8u72-jre.tar | docker load

Create Eclipse Project Metadata

$ ./gradlew eclipse

Import... Gradle Project...
From IDE to Docker Container

- Create Plan file
- Create Dockerizor instructions
<plan name="game-of-life" version="0.1" scoped="false" atomic="true"/>

<artifact type="bundle" name="com.eclipsesource.examples.gol.api" version="[0.1, 1)" />
<artifact type="bundle" name="com.eclipsesource.examples.gol.jenova" version="[0.1, 1)" />
<artifact type="bundle" name="com.eclipsesource.examples.gol.engine" version="[0.1, 1)" />
<artifact type="bundle" name="com.eclipsesource.examples.gol.commands" version="[0.1, 1)" />
<artifact type="bundle" name="com.eclipsesource.examples.gol.server" version="[0.1, 1)" />
<artifact type="bundle" name="com.eclipsesource.examples.gol.client" version="[0.1, 1)" />

</plan>

Plans encapsulate the artifacts of a Virgo application as a single unit.
dockerizor

./gradlew dockerize

dockerizor {
    repository = 'eclipsesource/virgo-tomcat-runtime'
    description = 'Virgo Server for Apache Tomcat'
    virgoFlavour = 'VTS'
}

Gradle Plugin Dockerizor

developed at GitHub https://github.com/eclipsesource/dockerizor
available from Gradle Plugins https://plugins.gradle.org/plugin/com.eclipsesource.dockerizor
```groovy
dockerizor {
    repository = 'game-of-life/runtime-only'
    javaImage = 'java:openjdk-8u72-jre'
    hudsonJobName = '3.7.0.M02'
    createLocalCopy = true
    removeAdminConsole = false

    postDockerizeHook = { task ->
        project.logger.info "Adding nashorn packages to configuration/java-server.profile"
        task.RUN "sed -i 's/org.xml.sax.helpers/org.xml.sax.helpers,\n jdk.nashorn.api.scripting/' ${project.dockerizor.virgoHome}/configuration/java-server.profile"
        task.RUN "sed -i 's/ sun.*/ sun.*,\n jdk.*/' ${project.dockerizor.virgoHome}/configuration/java-server.profile"
    }
}
```

dependencies {
  endorsed files('libs/nashorn.jar')

  repositoryExt 'com.fasterxml.jackson.core:jackson-core:2.6.4'
  repositoryExt 'com.fasterxml.jackson.core:jackson-annotations:2.6.3'
  repositoryExt 'com.fasterxml.jackson.core:jackson-databind:2.6.4'
}

gradle :runtime-only:dockerize

Adds 3rd party dependencies to ${VIRGO_HOME}/endorsed/libs

Adds 3rd party dependencies to ${VIRGO_HOME}/repository/ext
gradle :app:dockerize

dockerizor {
  ...
  pickupFiles = ['game-of-life.plan']
  dryRun = true
}

dependencies {
  ...
  repositoryUsr project(':game-api')
  repositoryUsr project(':jenova')
  ...
}
Task 7: Docker Deployment

07.1 search for official Java 8 image at https://hub.docker.com

07.2 add all game-of-life bundles to repository

07.3 add game-of-life bundles to Virgo plan file
Verify Deployment
End Deployment

git diff task_07_docker_final
Bonus Deployment

osgi> plan list
osgi>

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>game-of-life</td>
<td>0.1.0</td>
<td>ACTIVE</td>
</tr>
</tbody>
</table>

Enable OSGi console

Create and run a local copy of “Game-of-Life”
Congratulations, you made it!
Thank you!
Evaluate the Sessions

Sign in and vote at eclipsecon.org or use our EclipseCon App

-1 0 +1
Standard Shell Commands

lb  list bundles, use -s to see symbolic names
inspect capability service <bundle id>
  show all services provided by a bundle
start/stop
  start and stop bundles
grep  same as Unix command (use with pipe | )
headers print bundle headers
Virgo Shell Commands

`clhas`  Lists all bundles that **contain** a class or resource.

`clload` Lists all bundles that can **load** a class.

`plan list` Lists all plans.

Game of Life - Custom Commands

init \([x[,y]]\)  initialize a game
print                print the current board
run  \([ms]\)      run the game at the given speed
pause               pause the game
next                 calculate the next generation
toggle               toggle state (alive or dead) of a cell
Gradle Build Commands

./gradlew <task1> <task2>

- build: build project
- test: run the tests
- run: deploy the OSGi bundles via JMX
- dockerize: create Docker image

-x <task> skip a task
Docker Commands

docker

build Build a new image
run Create a new container and start it

[build] https://docs.docker.com/engine/reference/commandline/build/
[run] https://docs.docker.com/engine/reference/run/