OSGi™ Community Event 2010

jax 2010 LONDON

Dependencies, dependencies, dependencies

Marcel Offermans

luminis TECHNOLOGIES

Marcel Offermans

- Fellow and Software Architect at Luminis Technologies marcel.offermans@luminis.nl

- Member and Committer at Apache Software Foundation marrs@apache.org

Agenda

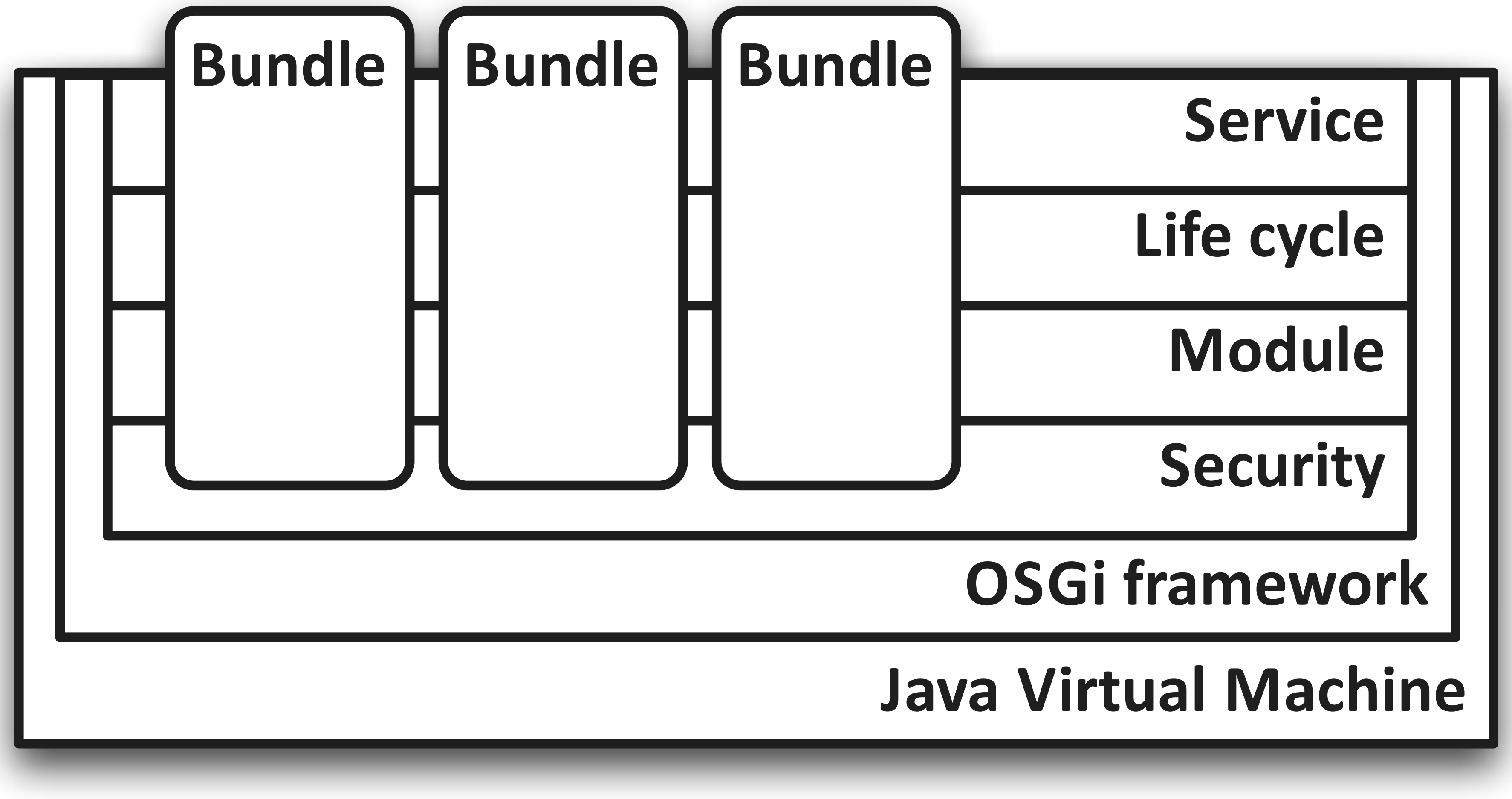
Agenda

Software distribution:
copy both zip archives from the memory stick

- Introduction
- Basic Concepts
- Dependencies
- Design Patterns
- Custom Dependencies
- Add-ons
- Wrap-up

Introduction

Framework


Bundle
Bundle
Bundle
Service
Life cycle
Module
Security
OSGi framework
Java Virtual Machine

Service Dependencies

- framework: ServiceListener
  - notification whenever there is a change in the service registry
  - but: only changes that occur while you are listening
- utility: ServiceTracker
  - solves the listening issue
  - adds ability to customize services

Problem

- using the framework supplied tooling, you are dealing with dependencies at a very low level
- a lot of boiler plate code is needed to implement real world scenarios
- there are very few design patterns that deal with composing an application based on services

Declaring your dependencies

- Service Binder
- Dependency Manager
- Declarative Services
- iPOJO
- Blueprint
- ...many more

Dependency Manager

- Subproject of Apache Felix
- Nearing a 3.0 release
- Some interesting new features

...I’m not unbiased, being it’s author

Basic Concepts

Basic Concepts

- Component: class that implements certain behaviour (a POJO)
- Dependency: something our component needs or wants (ie. a service)
- Service: OSGi service interface(s) under which our component is registered

Declarative API

- Declarative ≠ XML
- Using a Java API has many advantages:
  - less error prone because of syntax and compile time checking
  - refactoring support, completion
  - very readable through fluid API
  - everything in one place, no magic

Example code

- Projects
  - Download: http://www.xs4all.nl/~mfo/projects.zip
- Uses Eclipse + BndTools
  - Homepage: http://njbartlett.name/bndtools.html
  - Update Site: http://bndtools-updates.s3.amazonaws.com
- Based on a snapshot release of the upcoming Dependency Manager 3.0
- During the presentation, we will switch between slides and Eclipse to show running examples

Example c

On the memory stick:

a) an Eclipse update site for BndTools
b) a set of Eclipse projects

- Projects
  - Download: http://www.xs4all.nl/~mfo/projects.zip
- Uses Eclipse + BndTools
  - Homepage: http://njbartlett.name/bndtools.html
  - Update Site: http://bndtools-updates.s3.amazonaws.com
- Based on a snapshot release of the upcoming Dependency Manager 3.0
- During the presentation, we will switch between slides and Eclipse to show running examples

Using the Dependency Manager

```
Import-Package = org.apache.felix.dm;version="[3.0,4)"
```

```
public class Activator extends DependencyActivatorBase {
  public void init(BundleContext context, DependencyManager manager) throws Exception {
    manager.add(createComponent()
      .setImplementation(new HelloWorld())
    );
  }

  public void destroy(BundleContext context, DependencyManager manager) throws Exception {
  }
}
```

Basic Use Cases

- Declare a component

```
HelloWorld helloWorld = new HelloWorld();
manager.add(createComponent()
    .setImplementation(helloWorld)
);
```

- Declare it lazily

```
manager.add(createComponent()
    .setImplementation(HelloWorld.class)
);
```

Component Life Cycle

- methods of the component

```
public static class HelloWorldLifeCycle {
    private void init() { System.out.println("init"); }
    private void start() { System.out.println("start"); }
    private void stop() { System.out.println("stop"); }
    private void destroy() { System.out.println("destroy"); }
}
```

- setCallbacks(“init”, “start”, ...)
  setCallbacks(inst, “init”, “start”, ...)
  - to invoke the methods on ‘inst’
- ComponentStateListener
  - if you want to listen from the outside

Declaring as a service

- setInterface(...)
  - allows you to declare multiple services
  - allows you to specify service properties

```
manager.add(createComponent()
    .setInterface(LogService.class.getName(),
             new Properties() {{ put(Constants.SERVICE_RANKING, 20); }})
    .setImplementation(MyLogService.class)
```

Declaring Dependencies

- Adding a dependency

```
manager.add(createComponent()
    .setImplementation(HelloWorldLogger.class)
    .add(createServiceDependency()
        .setService(LogService.class)
    )
);
```

- Injects dependency
  - uses null object pattern
  - injects other “OSGi” instances
- setCallbacks(...)
  setCallbacks(inst, ...)

Dependencies

Dependencies

- Different types:
  - Service Dependencies
  - Configuration Dependencies
  - Bundle Dependencies
  - Resource Dependencies

Configuration Dependencies

- Based on Configuration Admin
  - designed for required dependencies
  - service.pid to identify the configuration
  - allows you to only accept *valid* configurations
    - update() throws ConfigurationException

Bundle Dependencies

- Depend on a bundle:
  - in certain states
  - with certain manifest entries
- Bundle instance can be injected

Resource Dependencies

- Resources are modeled as URLs
- Are provided by a repository
  - another bundle
  - an Eclipse workspace
  - some external source
- Filter on host, port, protocol, path and URL

Design Patterns

Design Patterns

- Moving up a level in abstraction
- OSGi is too low level to expose to everybody, but it is a great platform to build on
- Patterns provide a common language and describe solutions in context

Overview of Design Patterns

- Whiteboard Pattern
- Null Object Pattern
- “Singleton” Service
- Aspect Service
- Adapter Service
- Resource Adapter Service

Whiteboard Pattern

“don’t call us…

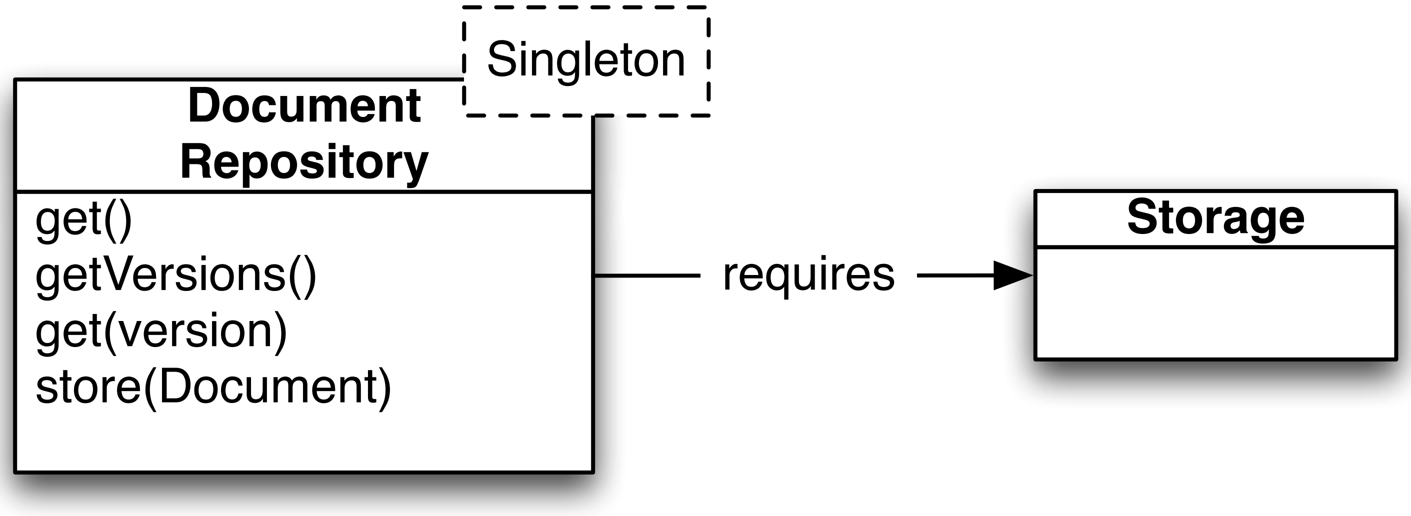
we’ll call you”

Null Object Pattern

- An object that implements a certain interface, can be safely invoked and does nothing

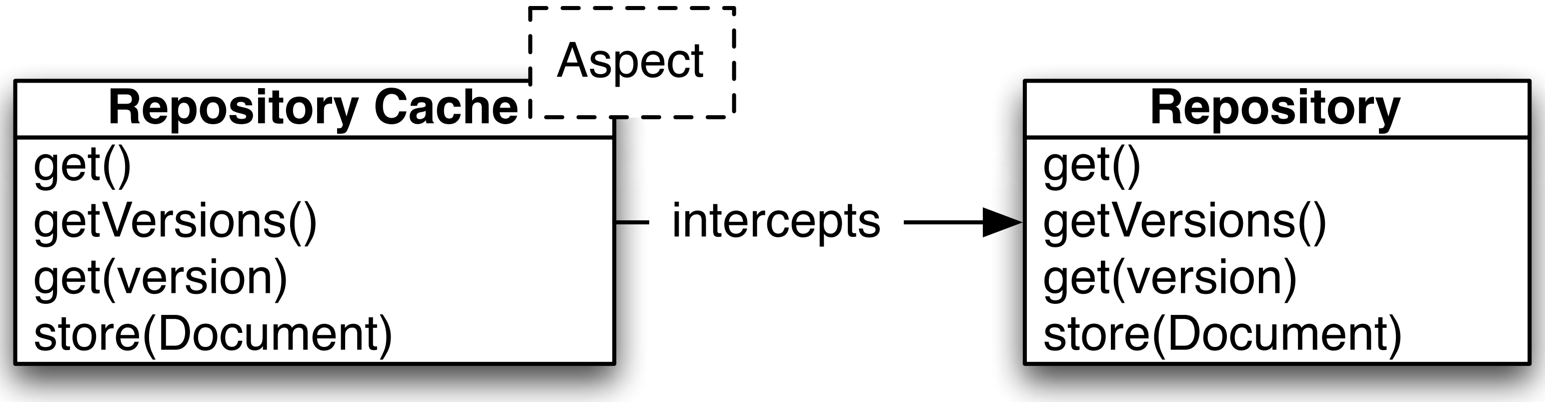
“Singleton” Service

- Publishes a component as a service
- Ties its life cycle to that of its dependencies



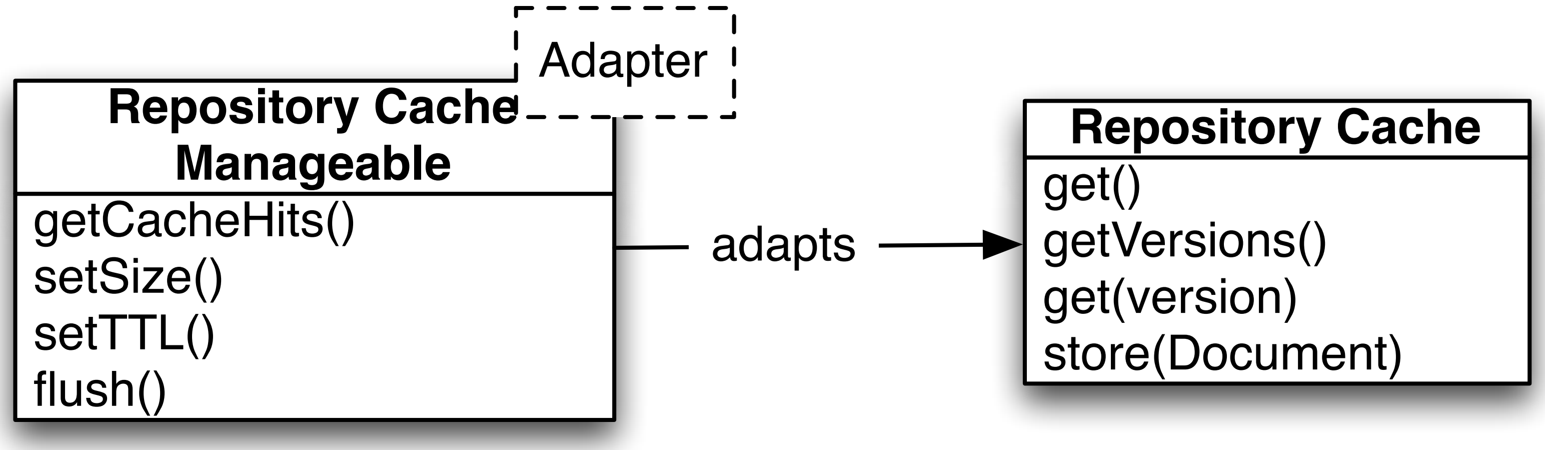
Aspect Service

- Works at the service level
- "intercepts" certain services
- can be chained based on rankings
- completely dynamic



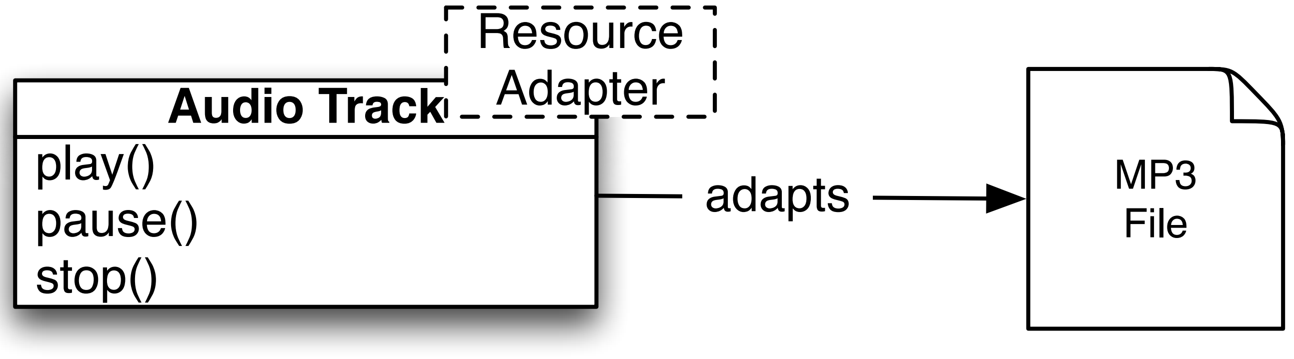
Adapter Service

- Works on existing services
- Adapts them, publishes a different service for each existing one



Resource Adapter Service

- Adapts resources (instead of services)
- Allows you to expose the behavior of certain resources instead of their "inner guts"



Custom Dependencies

Custom Dependencies

- Dependency Manager is extensible with custom dependencies:
  - depend on time of day
  - depend on some custom instance / condition (start level, app state)

Add-ons

Add-ons

- Shell (Felix, Equinox, Gogo)
- Annotation based (Java 5 required)
- Legacy support (2.x API adapter)

Wrap-up

Wrap-up

- Points to take away:
  - do not expose every developer to the OSGi API
  - build higher level abstractions, use design patterns
  - consider the dependency manager, it is very flexible

More about OSGi

- ApacheCon 2010 North America
  November 1–5, Atlanta
  - OSGi tutorial
  - full day OSGi track
- Masterclass on OSGi
  October 12–15, Girona
  - Neil Bartlett and Peter Kriens