



The Hundred Kilobytes Kernel (HK2)

Rikard Thulin & Ferid Sabanovic
IBS JavaSolutions



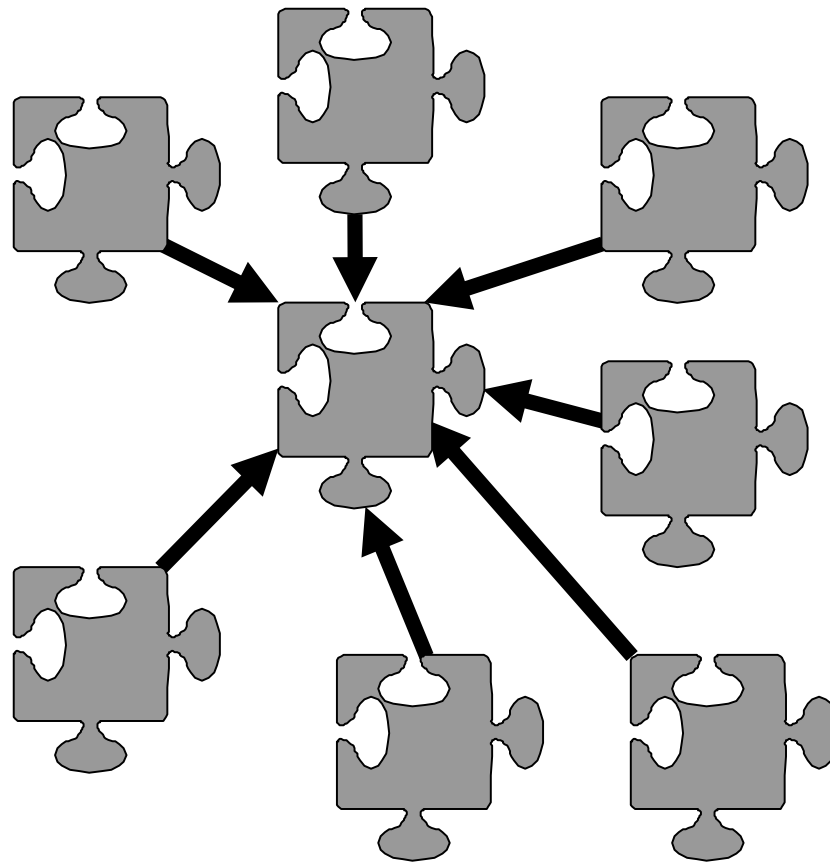
What we will present

- **A component based architecture**
 - Exemplified by explaining the details of one such implementation: HK2
 - ...could be realized by many others such as OSGi



The problem

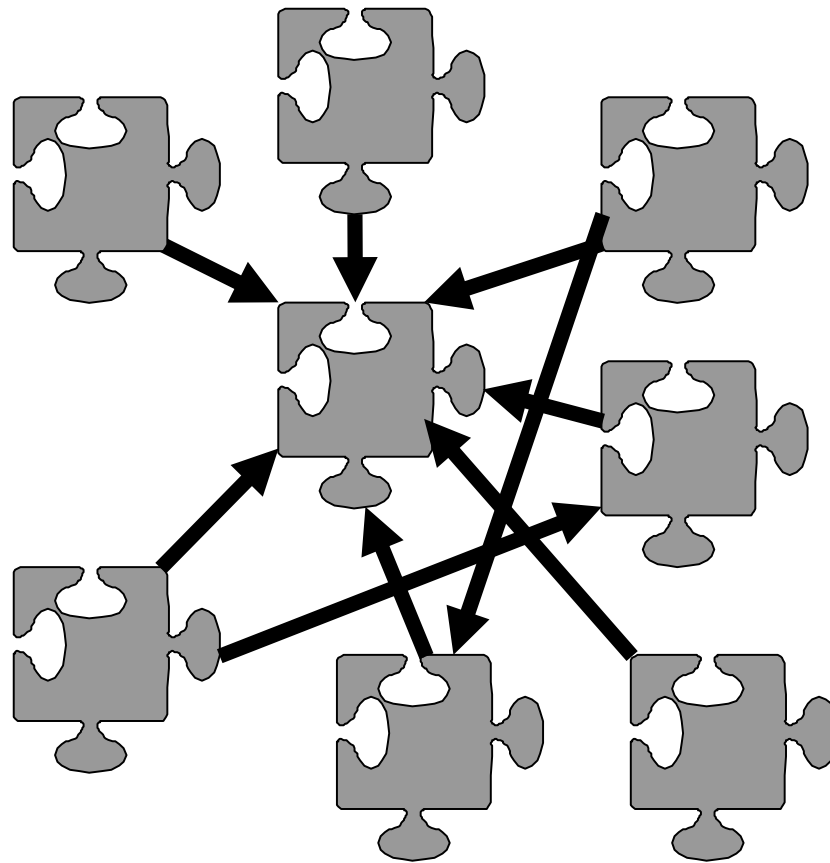
Version 1.0 with nice design





The problem

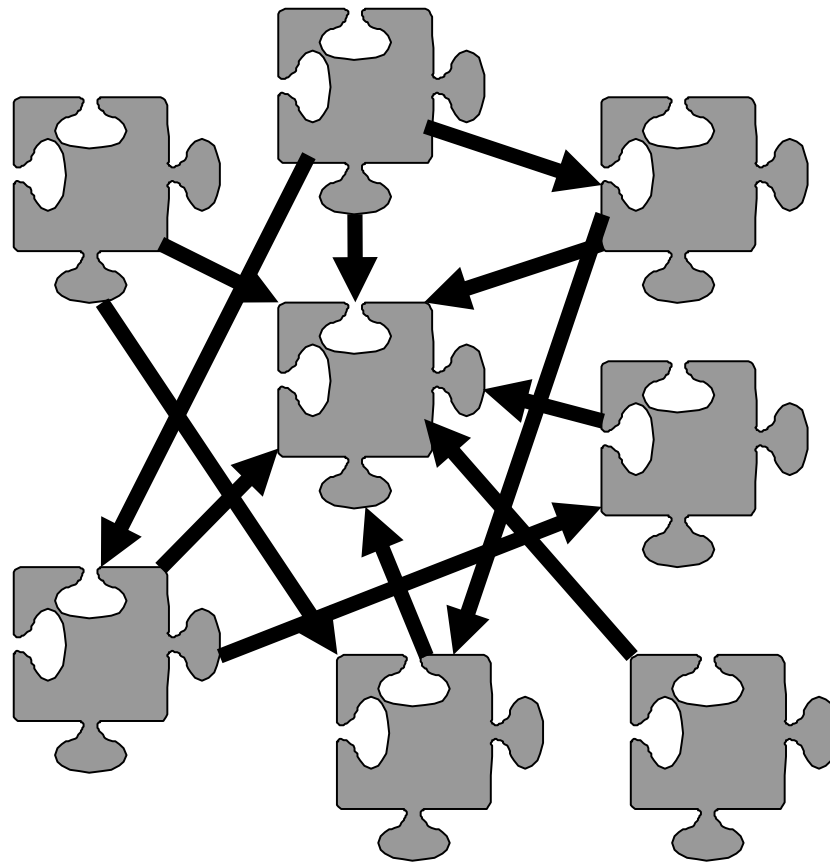
Version 1.1 – just needed a few hacks





The problem

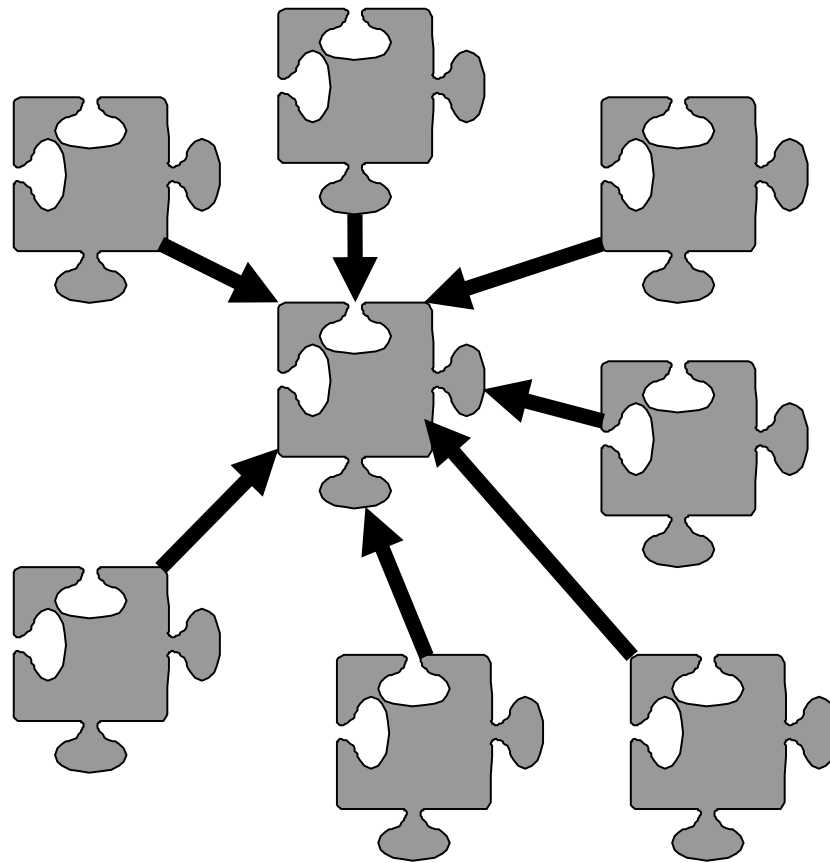
Version 2.0 – still works but...





The problem

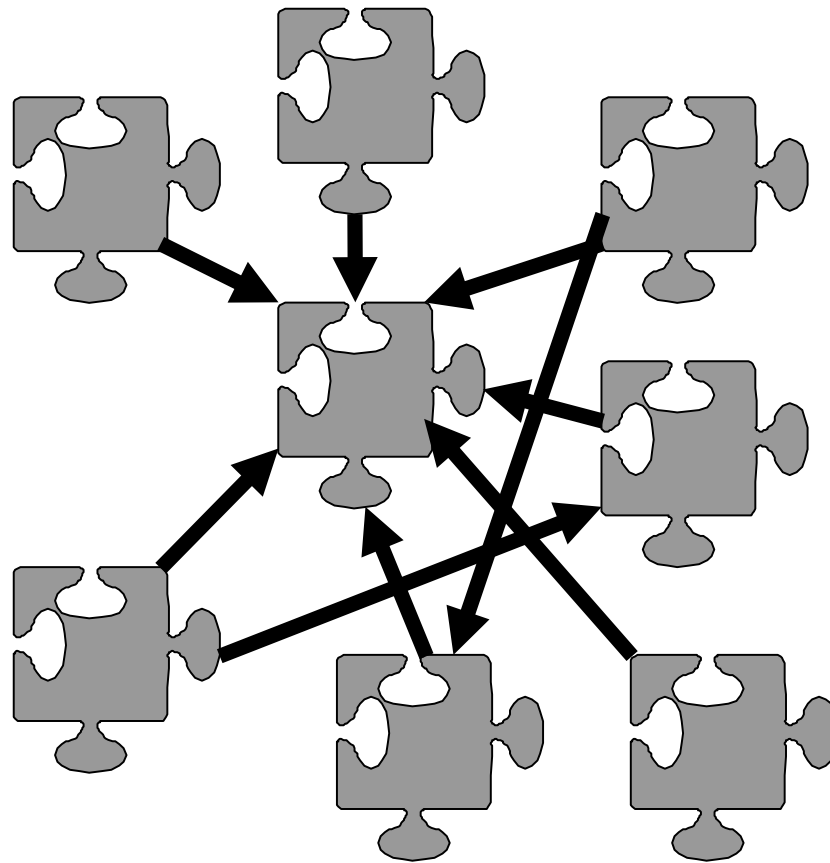
Version 3.0 – re-write, looking good again





The problem

Version 3.1 – just needed a few hacks...





And another problem

- **Deployment**

- We do not want to re-deploy the whole application, only that logical unit that was modified
- With a component based architecture this can be very clean and neat
- The Java IDE:s does this a lot...

- **Releases Managment**

- Possible to release features when ready, avoiding big bang releases



The only historical slide

- **Component based architectures is nothing new**
 - Has been around for years
 - OSGi about 10 years
- **The interest picked up when Eclipse switched to OSGi**
- **All (or almost) application servers are going there**



HK2 from 10K meters

- **Micro kernel for applications**
- **HK2 is runtime container**
- **Applications are divided into modules**
 - Version
 - Dependencies
- **The HK2 runtime loads the module and resolves dependencies**
- **No Class-Path needed**



DEMO

”What it is”



HK2

- **Environment with encourages healthy design**
- ***“a module subsystem coupled with a simple yet powerful component model to build software”***
- **Based on contracts**
- **Separation of API and SPI**
- **Dynamic, components discovered at runtime**
- **Applications are “composed” in runtime**



HK2

”HK2 proposes a model which is aimed to be friendly to existing technologies such as OSGi yet will provide a path to the implementation of modules in Java SE 7”

There is a lot of politics regarding JSR 277 and OSGi. As we do not represent SUN or OSGi we will likely not be able to answer questions about this



We all love acronyms (WALA)

- **Many names:**
 - Service Oriented Architecture (SOA)
 - Service Component Architecture (SCA)
 - Module / Component System
- **Do you care for a...**
 - Service
 - Component
 - Module
 - Bundle
 - Plugin



HK2 terminology

- **Contract**
 - An Interface marked by an annotation
- **Service Implementation**
 - The class (source file) marked by an annotation that implements a Contract
- **Service Instance**
 - The runtime instance of some Service Implementation
- **Services are grouped in modules**
 - Jar file + manifest
- **Module Definition**
 - Meta information stored in the manifest



HK2 Facts

- **Loosely based on Java Module System (JSR 277)**
 - Branch exists to run on top of the JSR 277 implementation
- **Small footprint, ~50Kb**
 - Less than ~5000 LOC (without comments)
 - Impressive design: clean and elegant
- **Runs on Java SE 5**
- **It is the foundation for GlassFish V3**
- **License same as GFv3: GPLv2 and CDDL**



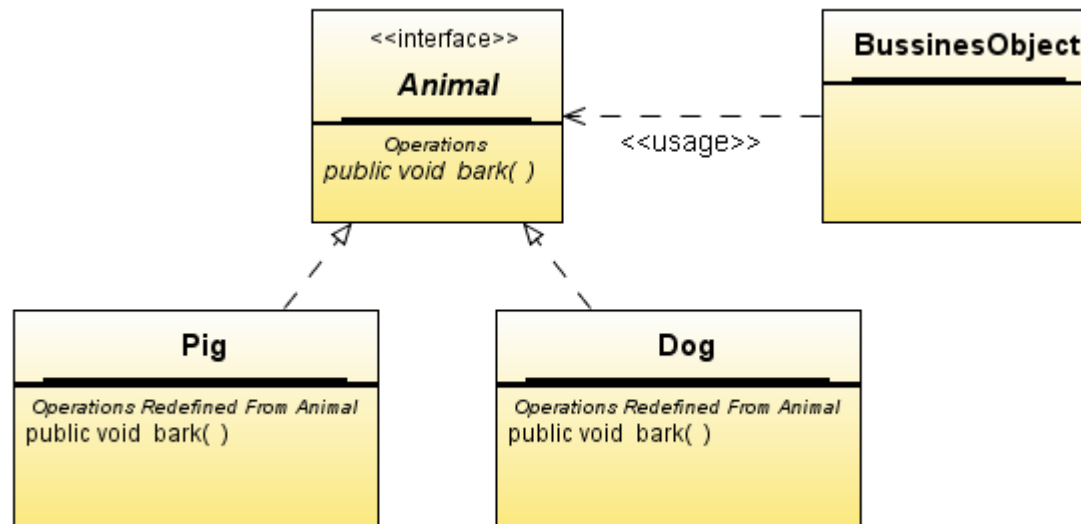
The two parts

- **The module subsystem**
 - Responsible for loading and unloading modules
 - 23 classes and 7 interfaces
- **The components runtime**
 - Create instances of Services
 - 16 classes and 5 interfaces



What we have

- A simple application packaged into a JAR-file
- Once interface (contract) and two classes implementing it
- One class implementing bussines logic using the contract





Modules

- **Modules are Plain Old Java Jarfiles**
 - Meta information stored in the manifest
- **Declare their dependencies to other modules**
- **Has a life cycle**
 - Dynamically loaded and unloaded
- **Allows multiple version at the same time**



Module definition

- **A module is defined by**
 - Name, "se.jsolutions.hk2.demo1"
 - Version number, "1.2.3-rc1"
 - Imports (dependencies), "se.jsolutions.hk2.demo2"
 - Exports (SPI), "se.jsolutions.hk2.demo1.spi"



Module definition: name

- **Any string but in reality the package name**
- **Must be unique** (for the universe and beyond)
- **Dependencies are declared by name**



Module definition: version

- **The format of a version is defined as:**

major.minor[.micro[_patch]][-qualifier]

major, minor and micro are non-negative integers

patch indicates a patch release

String that indicates a non FCS release

Example: 3.2-RC1
 3.2.1



Module definition: imports

- **A module may depend on 0 to n modules**
- **Modules are by default shared by its users**
 - Possible to do a private import
- **Imports can be limited by version range**
 - Open range: a.b+
 - Family range: a*



Module definition: re-exports

- **It is possible to re-export an imported module, kind of “module composition”**
 - Valuable for containers



Module packaging

- **Modules are packaged in a JAR file**
- **The manifest is used to describe the module**
 - Module-name
 - Module-version
 - Module-exports
 - etc

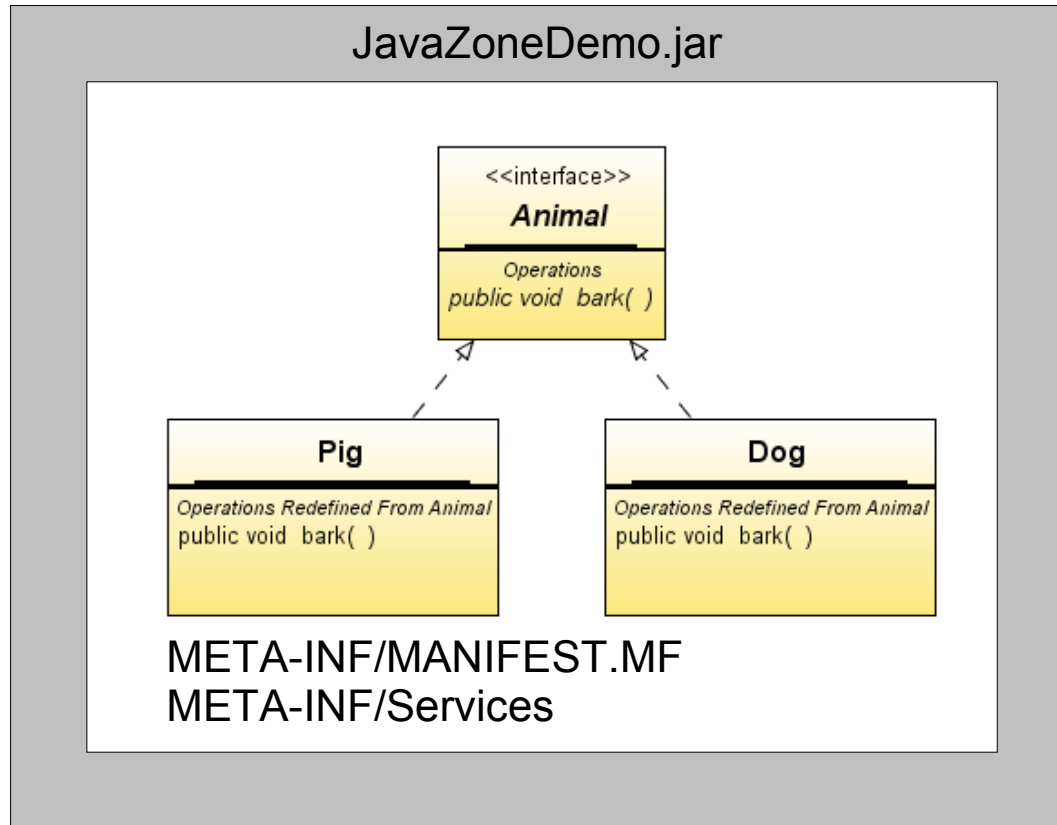


Creating a module

- **Meta data is added to the jar-file**
 - MANIFEST.MF
 - Name: se.javasolutions.hk2.demo2
 - Version number: 0.1-SNAPSHOT
 - No dependencies at this time
 - Services
 - The "pig" and "dog" service
- **A maven plugin does the work**



Creating a module





DEMO

”Creating a module”



Module initialization

- **A module can be in the following states:**
 - NEW
 - PREPARING
 - VALIDATING
 - RESOLVED
 - READY
 - ERROR
- **As a module developer you do not need to care**



Module unloading

- **GC does the job, thus modules can not be programmatically unloaded**
- **A module can be unloaded when**
 - No other module has dependencies to it
 - All instances of all classes has been GC
 - The modules is not defined as “sticky”
- **Removed from the registry**



ClassLoader

- **To enforce the module contract only the public interface is visible to external user**
- **Two ClassLoaders are used**
 - Public façade ClassLoader
 - Private ClassLoader
- **The module subsystem can bootstrap itself**
- **No more classpath**
 - `java -jar javazone_rocks_demo.jar`



Module repository

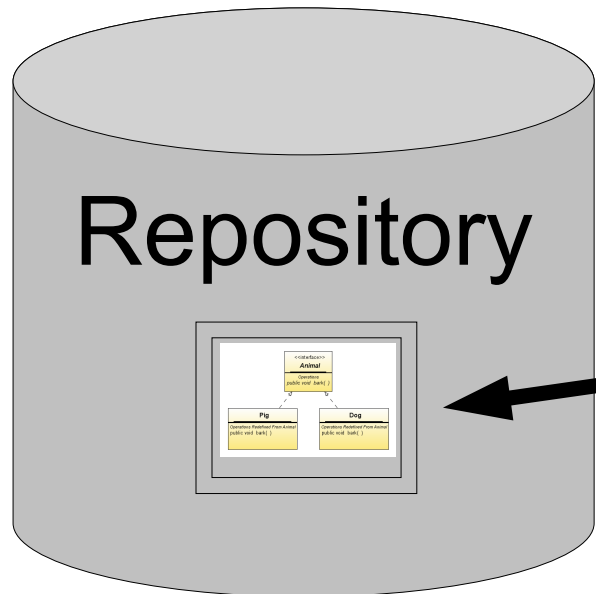
- **Storage for modules**
 - Local or remote
 - Has a weight
- **Modules can be added and removed in runtime**
- **Different implementations**
 - Disk based
 - Maven 2 repository
 - JSR 277 *
 - Or create your own JavaSpace repository...

* not implemented



The module repository

- **The module is placed into a repository**
 - We use a directory based repository



← The "demo" module (jar)



DEMO

”Repository”



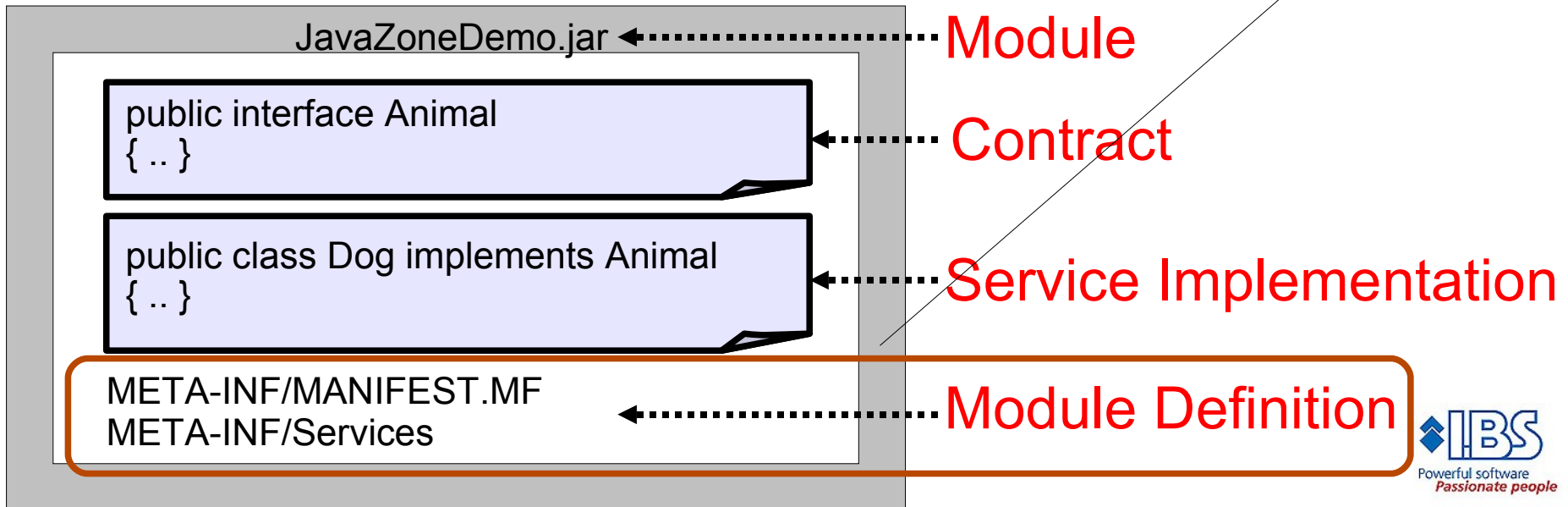
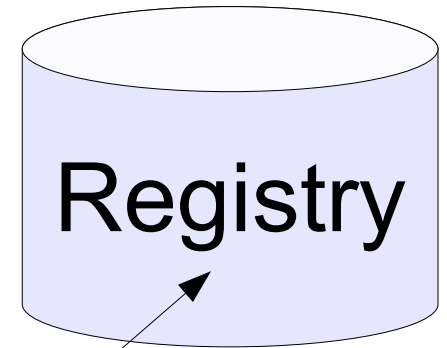
Modules Registry

- **Container for modules instances**
- **Only one shared instance of a module in one class loader**
 - Be aware of private imports...



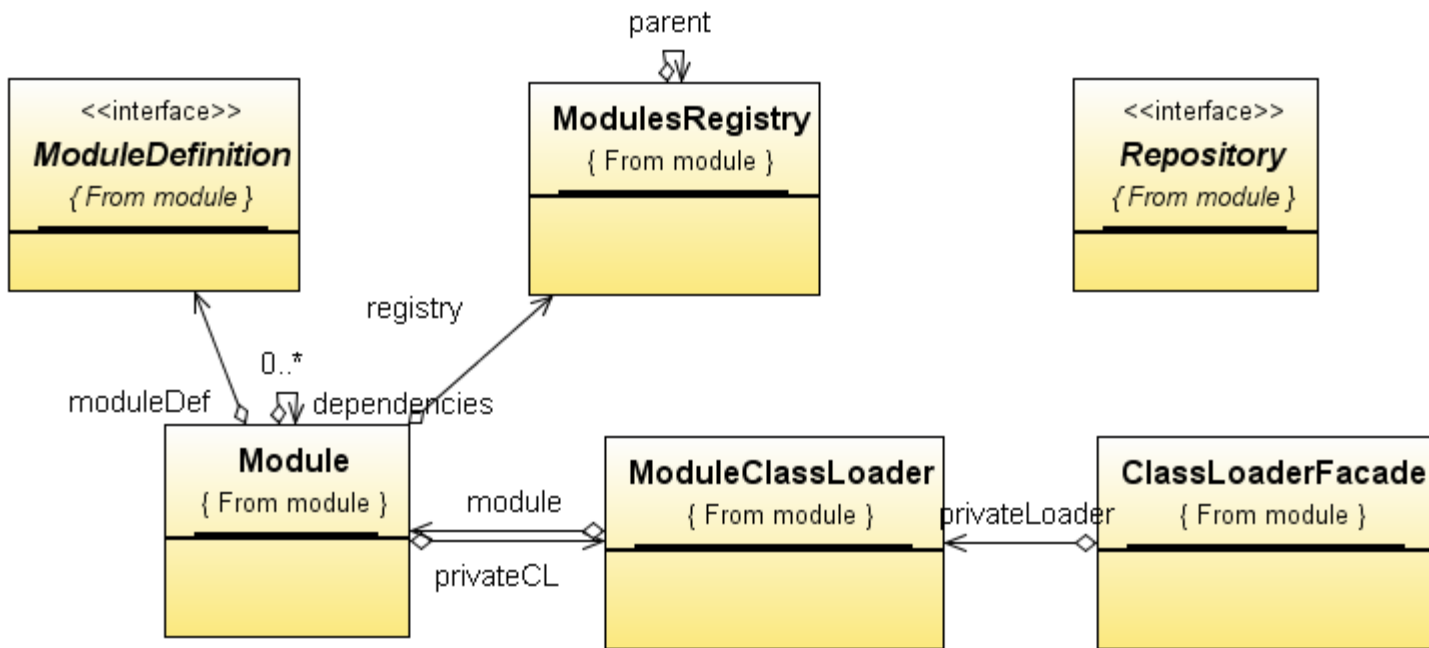
The Modules Registry

- The module instance is the runtime equivalent of the module definition





Modules Class Diagram





HK2 Components Runtime

- **Creates and configures objects**
 - Injecting required objects and its configuration
 - Makes objects available so it can be injected by others

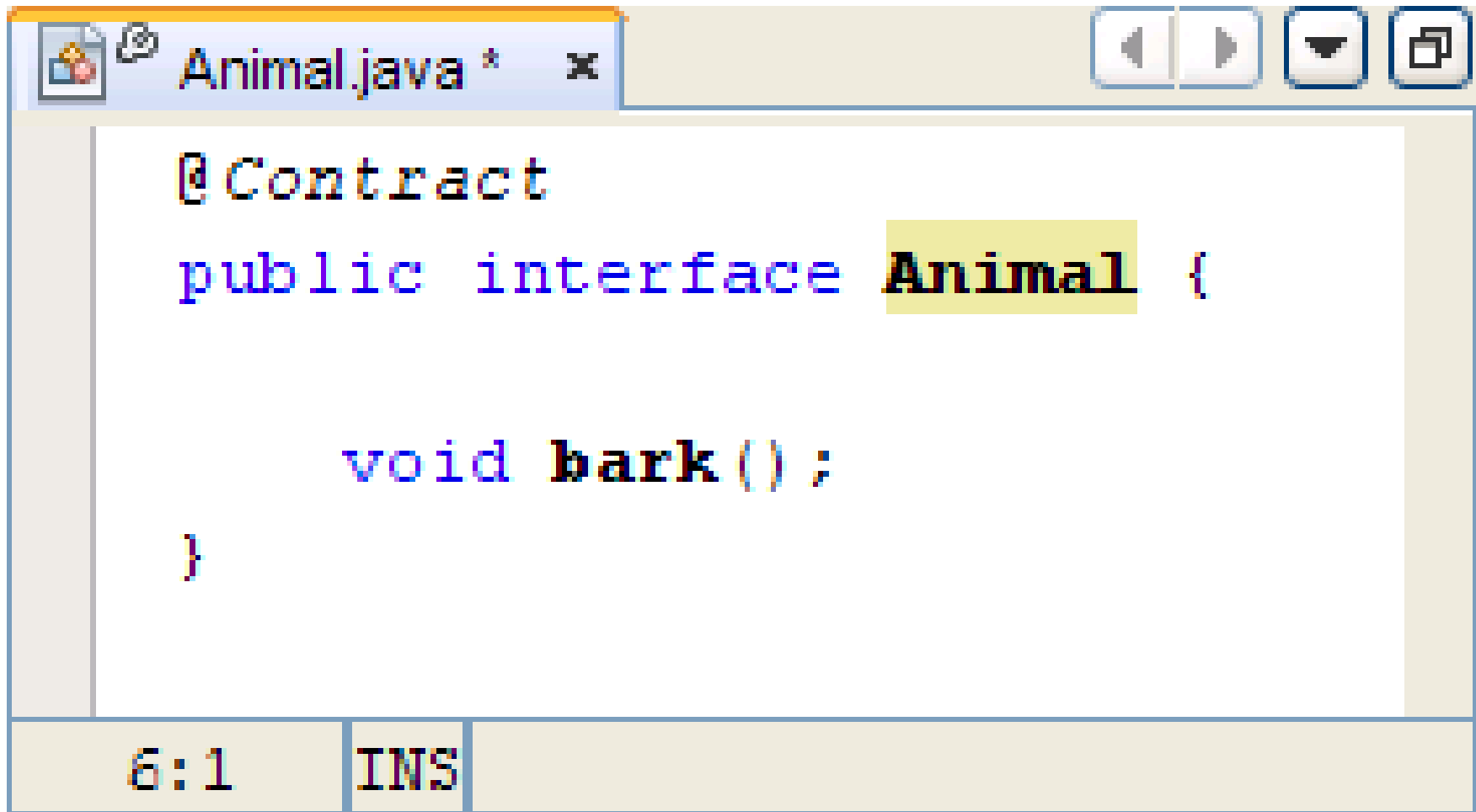


Services

- **POJO**
- **Identifies the building blocks or the extension points**
 - Example: GF3 admin commands
- **State-full or state-less**
- **Declared by META-INF/services in the jar**
- **Two annotations**
 - @Contract
 - @Service



Annotations



```
@Contract
public interface Animal {

    void bark();

}
```

6:1 INS



Annotations

```
Dog.java x [Navigation icons]
@Service
public class Dog implements Animal {
    public void bark() {
        System.err.println("Wow, wow!");
    }
}
```



Scope

- **Services instances has a scope**
 - Singleton
 - Per thread
 - Per application
 - Or custom...
 - GridScope / RemoteScope
 - PooledScope
- **Scopes are Services themselves**
- **A scope is responsible for storing the service instance tied to itself**



DEMO

”Scope”



Instantiation

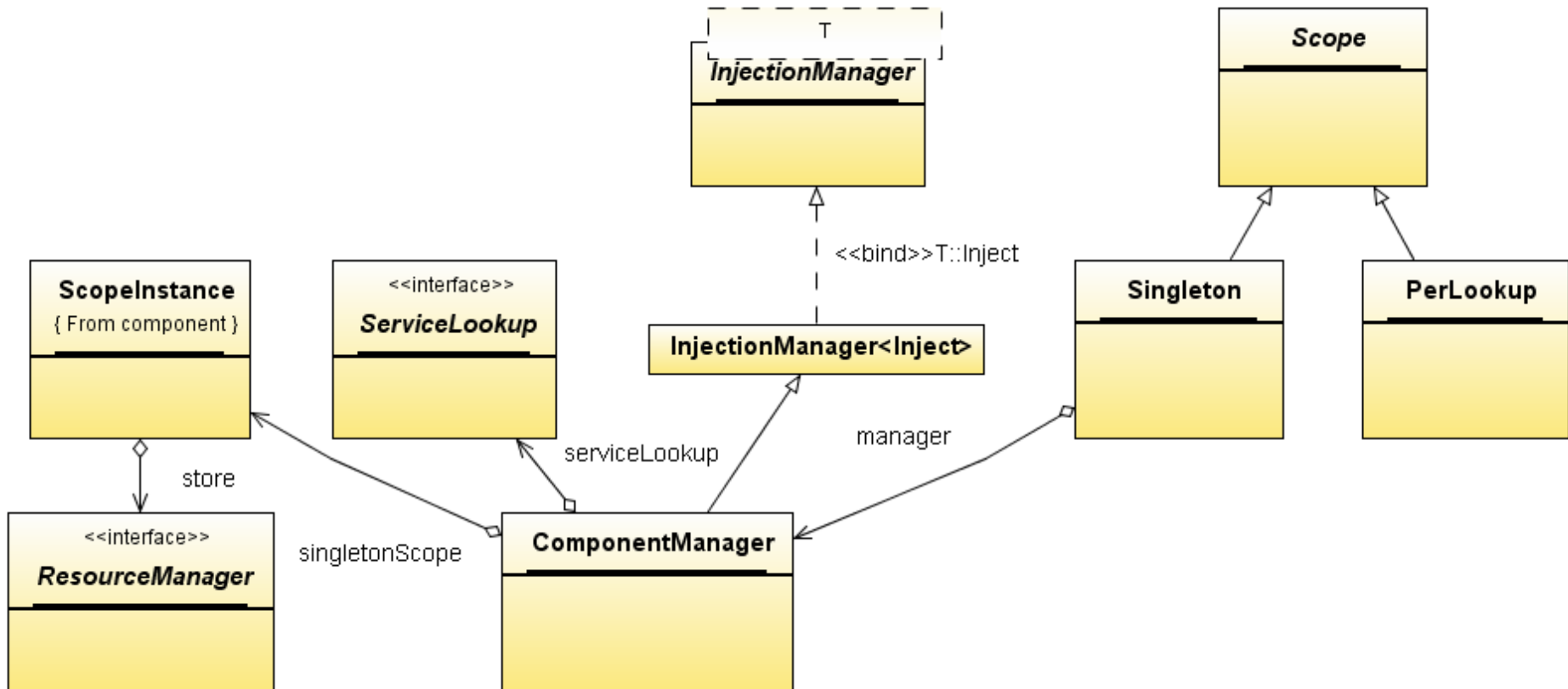
- **Instances are created by the ComponentManager**
 - "new" is never used
- **Instances can be injected (IoC) to fields or setters**
 - @Inject annotation
 - Instance can be further qualified with scope and name

```
@Inject(Scope=Singleton.class, name="Dog")  
Animal animal;
```

```
@Inject  
public void setAnimal(Animal animal) { ... }
```



Components Class Diagram





Other component / module subsystem technologies

- **OSGi**
 - Knopflerfish
 - Apache Felix
 - Eclipse Equinox
- **JINI & JXTA**
- **OpenWings**
- **Java Business Integration (JBI)**
- **Maven 2**
- **JSR 277**
 - Java Module System
- **JSR 294**
 - Improved Modularity Support in the Java Programming Language



References

- **Presentation and source will be available at**
<http://jsolutions.se>
- **HK2 web site**
<https://hk2.dev.java.net/>
- **JSR 277: Java Module System**
<http://jcp.org/en/jsr/detail?id=277>
- **OSGi™ - The Dynamic Module System for Java™**
<http://osgi.org/>



Q & A



About the authors

- **Consultants at IBS JavaSolutions**

- Rikard Thulin has been working with Java for over 10 years. In a previous life he worked as a Java Architect at Sun Microsystems Java Center. Rikard is a board member of the Swedish Java User Group “Javaforum.se”. Rikard holds a Master of Science in Software Engineering.
- Ferid Sabanovic interests include J2EE and other similar object oriented technologies like .NET. Ferid is actively involved in the Swedish Java User Group “Javaforum.se”. Ferid holds a B.Sc degree in Informatics.