



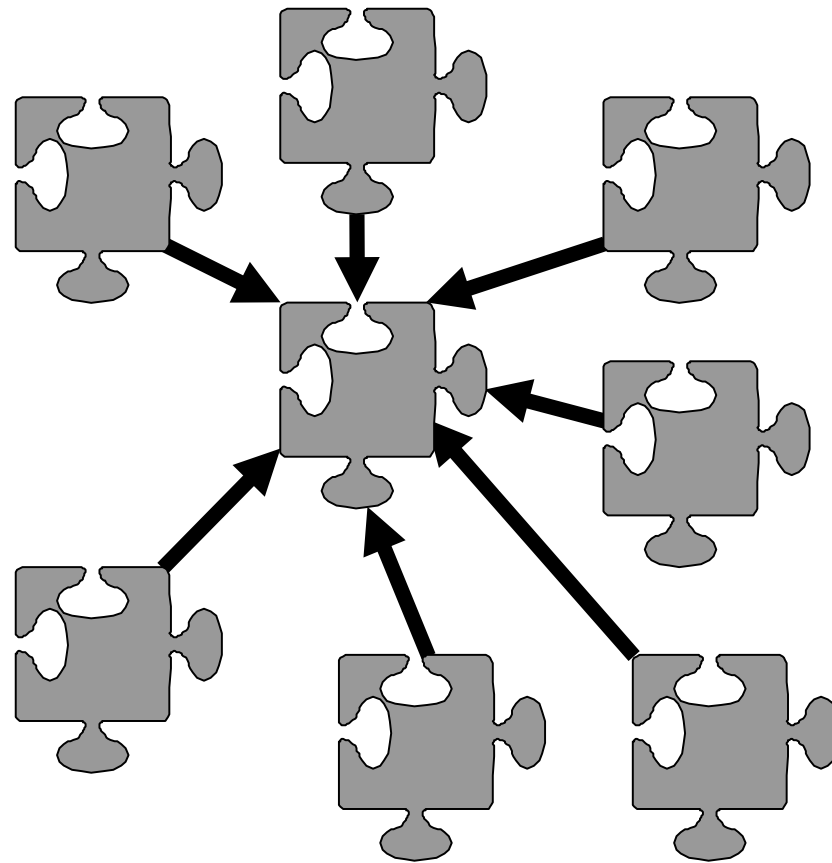
The Hundred Kilobytes Kernel (HK2)

Rikard Thulin & Ferid Sabanovic
IBS JavaSolutions



The problem

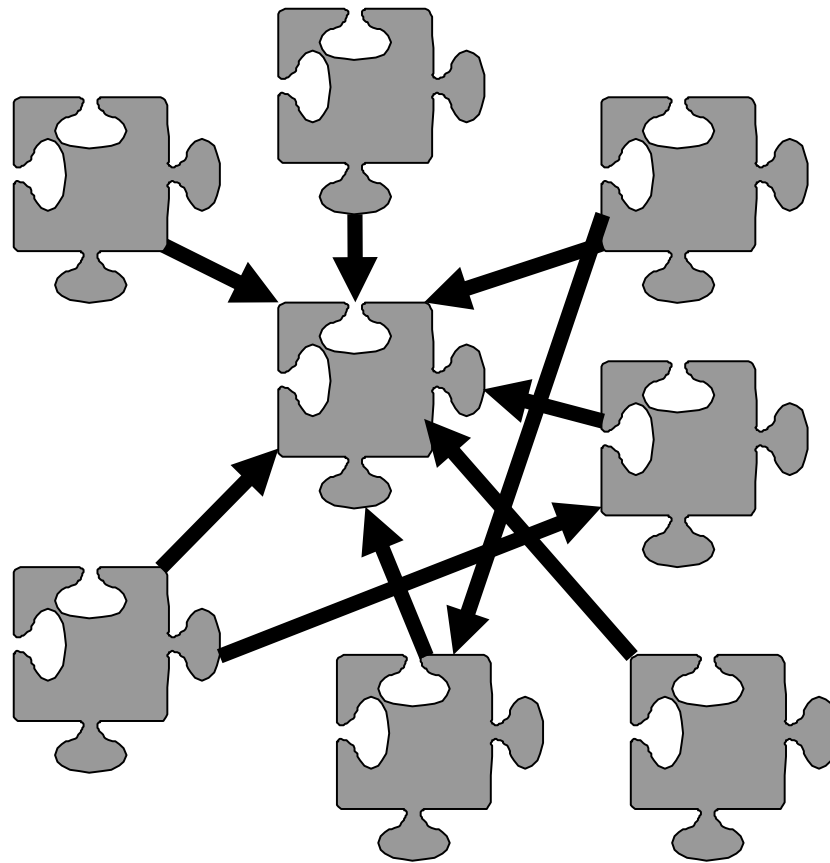
Version 1.0 with nice design





The problem

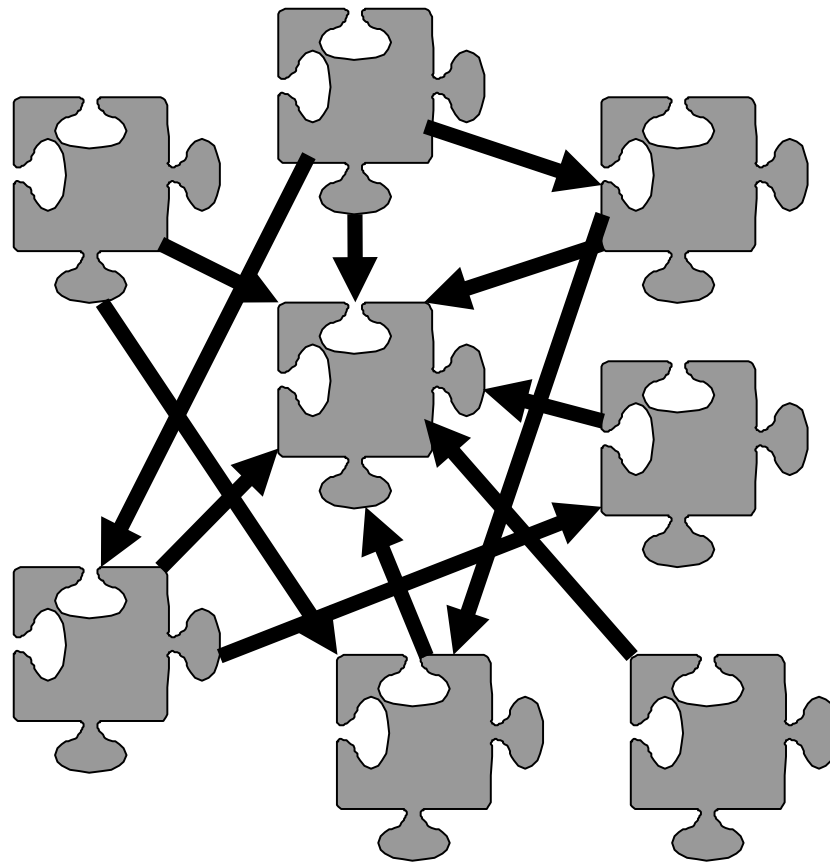
Version 1.1 – just needed a few hacks





The problem

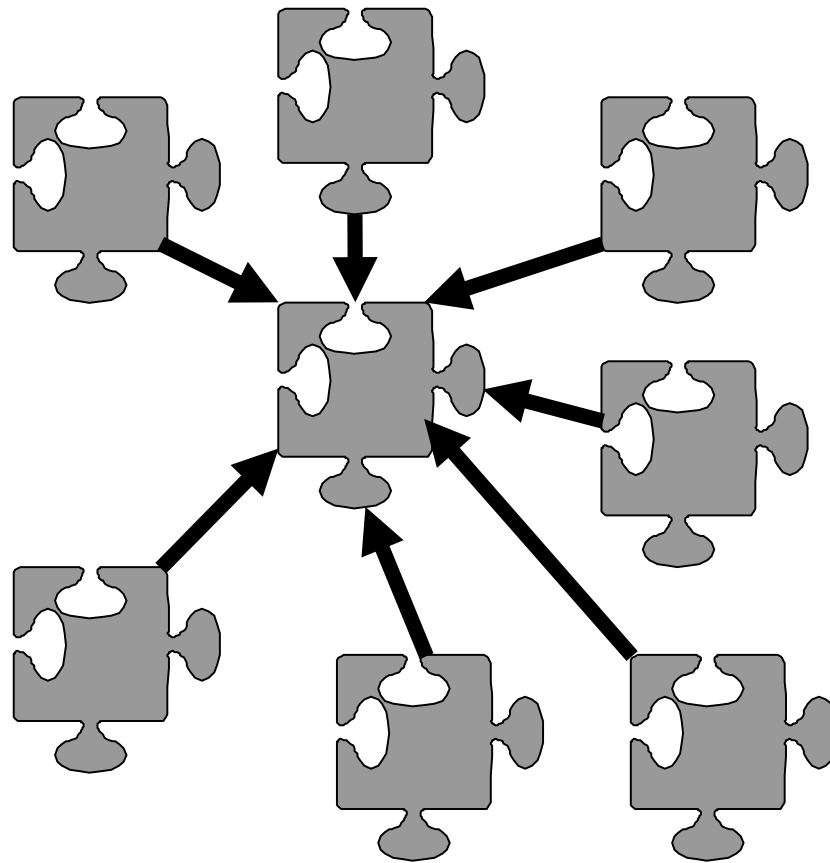
Version 2.0 – still works but...





The problem

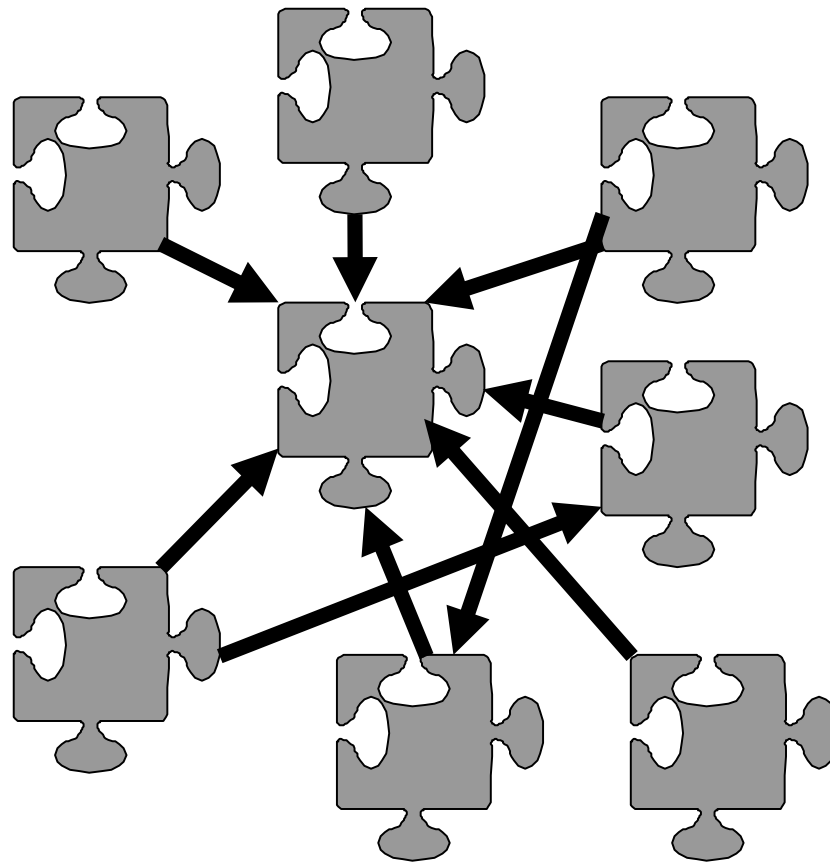
Version 3.0 – re-write, looking good again





The problem

Version 3.1 – just needed a few hacks...





HK2 from 10K meters

- **HK2 is runtime container**
- **Applications are divided into modules**
 - Version
 - Dependencies
- **The HK2 runtime loads the module and resolves dependencies before the applications main method is executed**



DEMO

”What it is”



HK2

- ***“a module subsystem coupled with a simple yet powerful component model to build software”***
- **Environment with encourages healthy design**
- **Requires a runtime container**
- **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”



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**
 - An instance of the modules meta data
- **Module Instance**
 - Represents a set of resources accessible for other modules



HK2 Facts

- **Loosely based on Java Module System (JSR 277)**
 - Is not the reference implementation for JSR 277
- **Small footprint, ~50Kb**
 - Less than ~5000 LOC (without comments)
- **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



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 a 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



Module Instances

- **A module instance is the runtime object module bundle (the JAR file)**
 - Dependencies to other modules
 - Exported APIs
 - The state of the module



DEMO

”Creating a module”



Module initialization

- **A module can be in the following states:**
 - NEW
 - PREPARING
 - VALIDATING
 - RESOLVED
 - READY
 - ERROR



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**



DEMO

”Module unloading”



ClassLoader

- **To enforce the module contract only the public interface is visible to external user**
- **Two ClassLoaders are used**
 - Public facade 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



DEMO

”Repository”

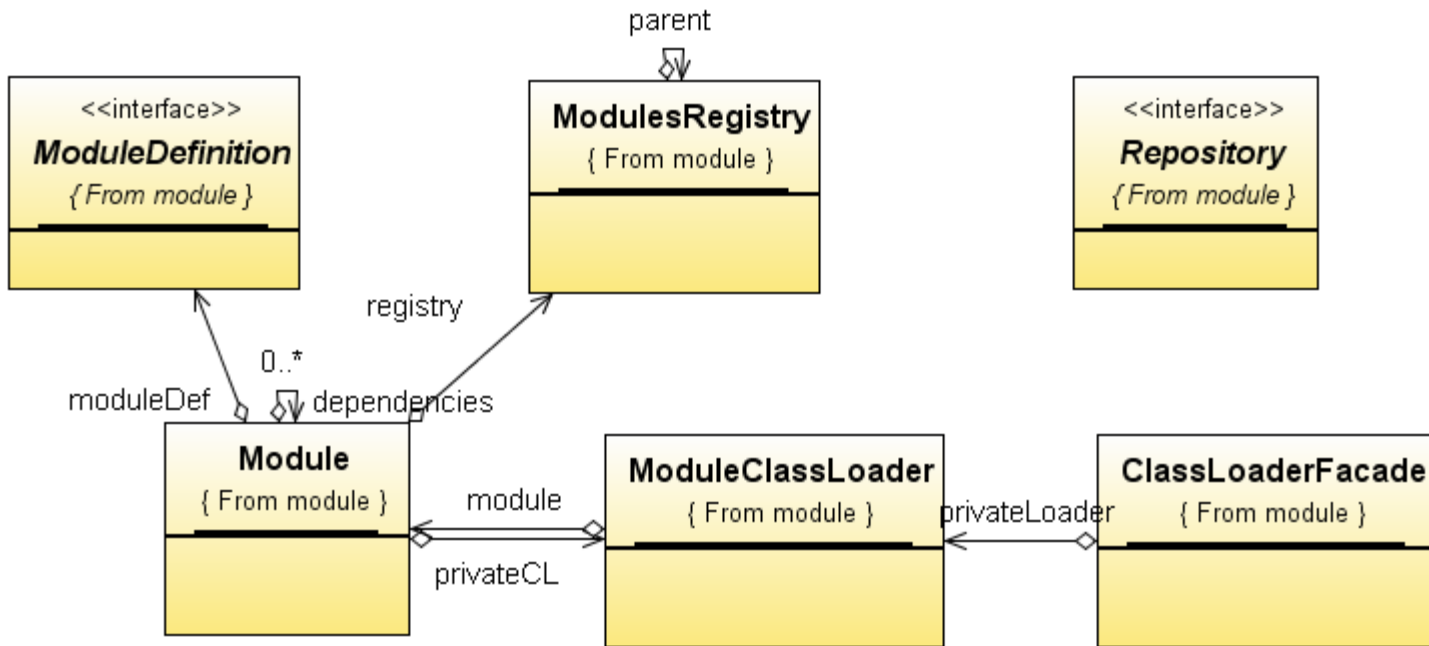


Modules Registry

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



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

- **Identifies the building blocks or the extension points**
- **POJO**
- **State-full or state-less**
- **Declared by META-INF/services in the jar**
 - Maven plugin
- **Two annotations**
 - @Contract
 - @Service



Services

@Contract

```
public interface HelloContract { ... }
```

@Services

```
public class HelloJavaZone implements HelloContract  
{...}
```



Scope

- **Services instances has a scope**
 - Singleton
 - Per thread
 - Per application
 - Or custom...
 - GridScope
 - DateScope
- **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="JavaZone")  
HelloService hello;
```

```
@Inject  
public void setHelloService(HelloService hello) { ... }
```



IoC: Extraction

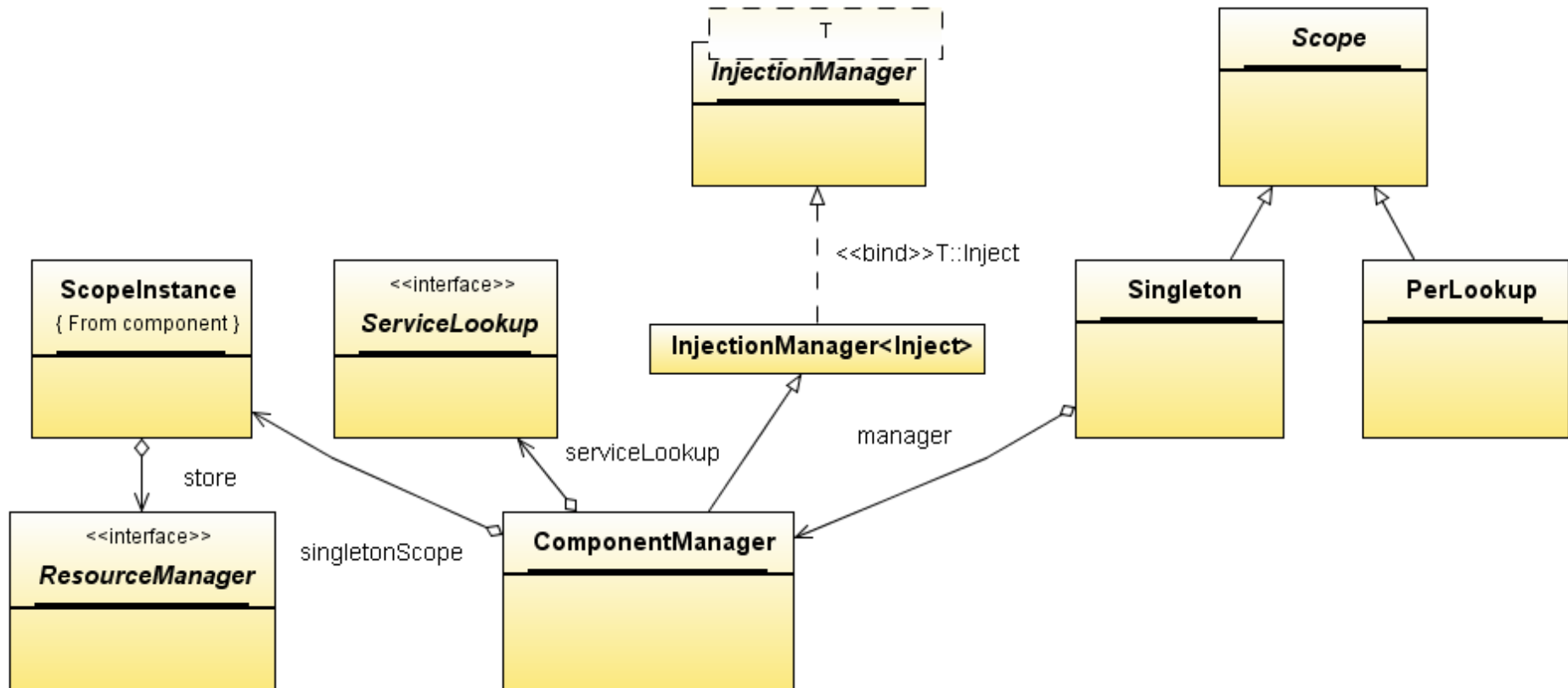
- **Services are automatically placed into its Scope**
- **More can be extracted from a Service**
 - @Extract annotation

```
@Extract  
HelloJavaZoneService hello;
```

```
@Extract  
public HelloJavaZoneService getHelloJavaZoneService() { ... }
```



Components Class Diagram





DEMO

“Putting the Pieces Together”



Other component / module subsystem technologies

- **OSGi**
 - Knopflerfish
 - Apache Felix
- **OpenSIS**
- **JINI & JXTA**
 - OpenWings
- **Java Business Integration (JBI)**
- **Maven 2**
- **JSR 277, JSR 294**



References

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



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 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. Ferid holds a B.Sc degree in Informatics.