The Hundred Kilobytes Kernel (HK2)

Rikard Thulin & Ferid Sabanovic
IBS JavaSolutions

{ firstname.surname@ibs.se }

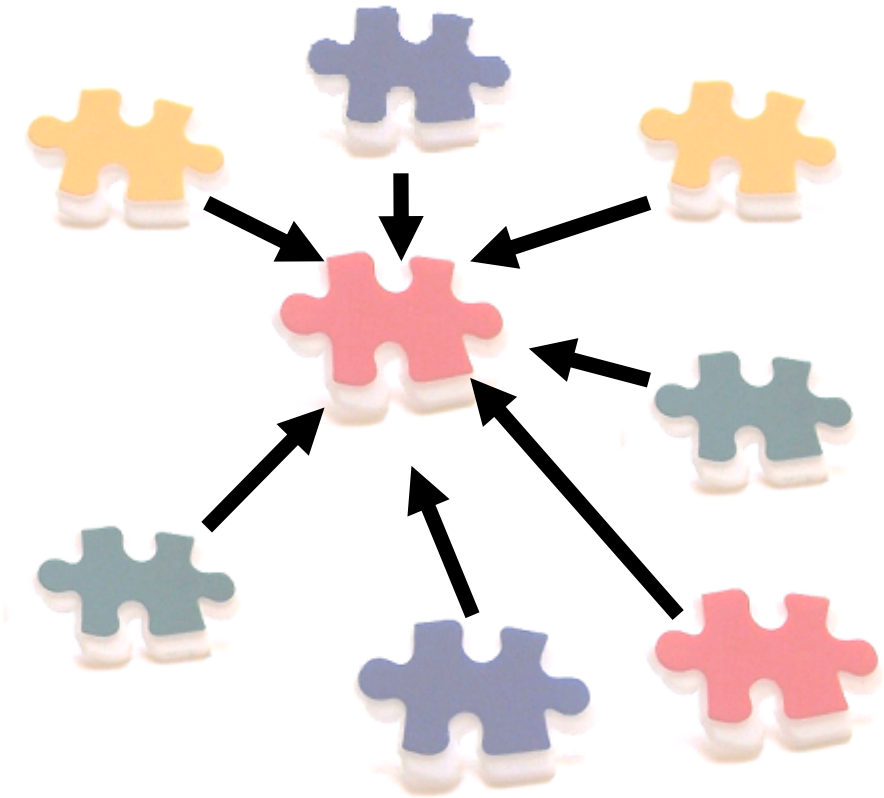
IBS
Powerful software
Passionate people

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
  - We will highlight some differences between OSGi and HK2

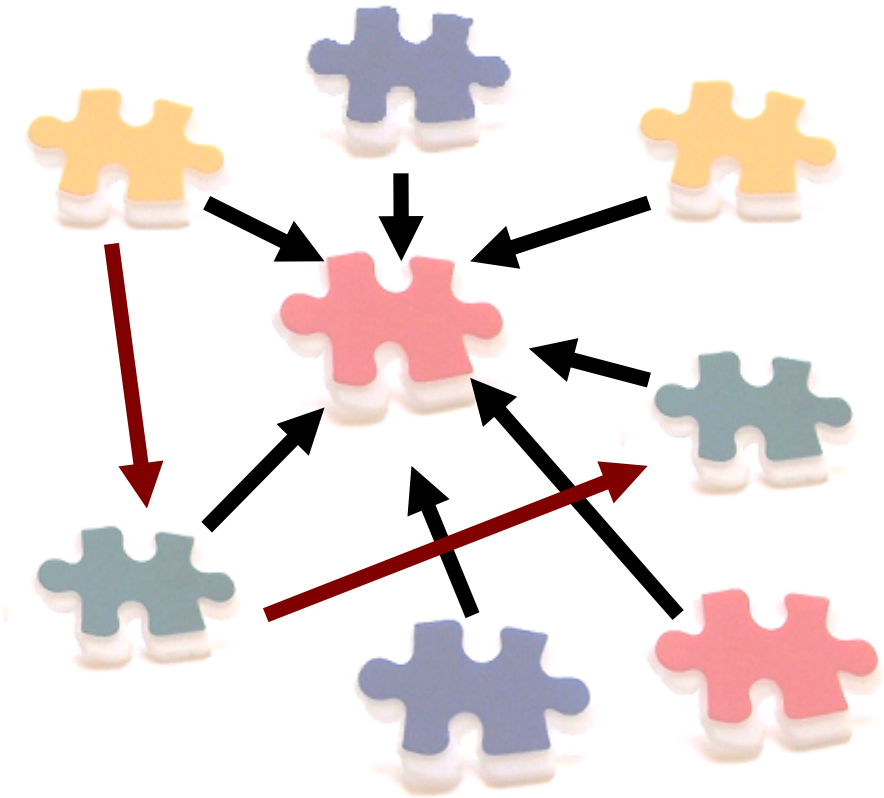
The problem

Version 1.0 with nice design



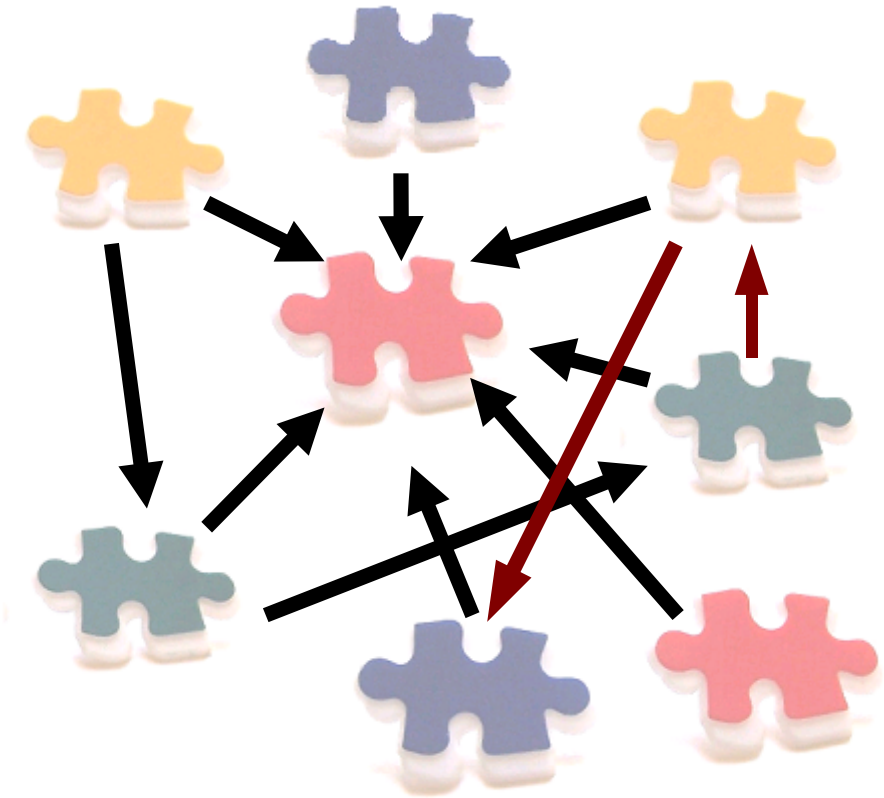
The problem

Version 1.1 – just needed a few hacks



The problem

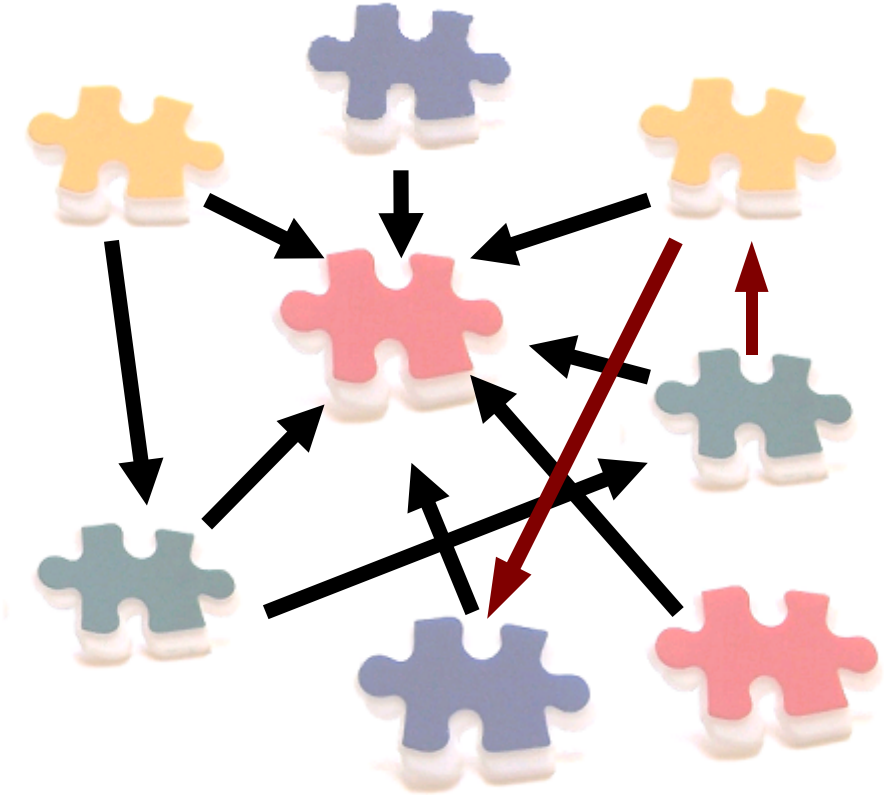
Version 1.2 – still works but but messy...



The problem

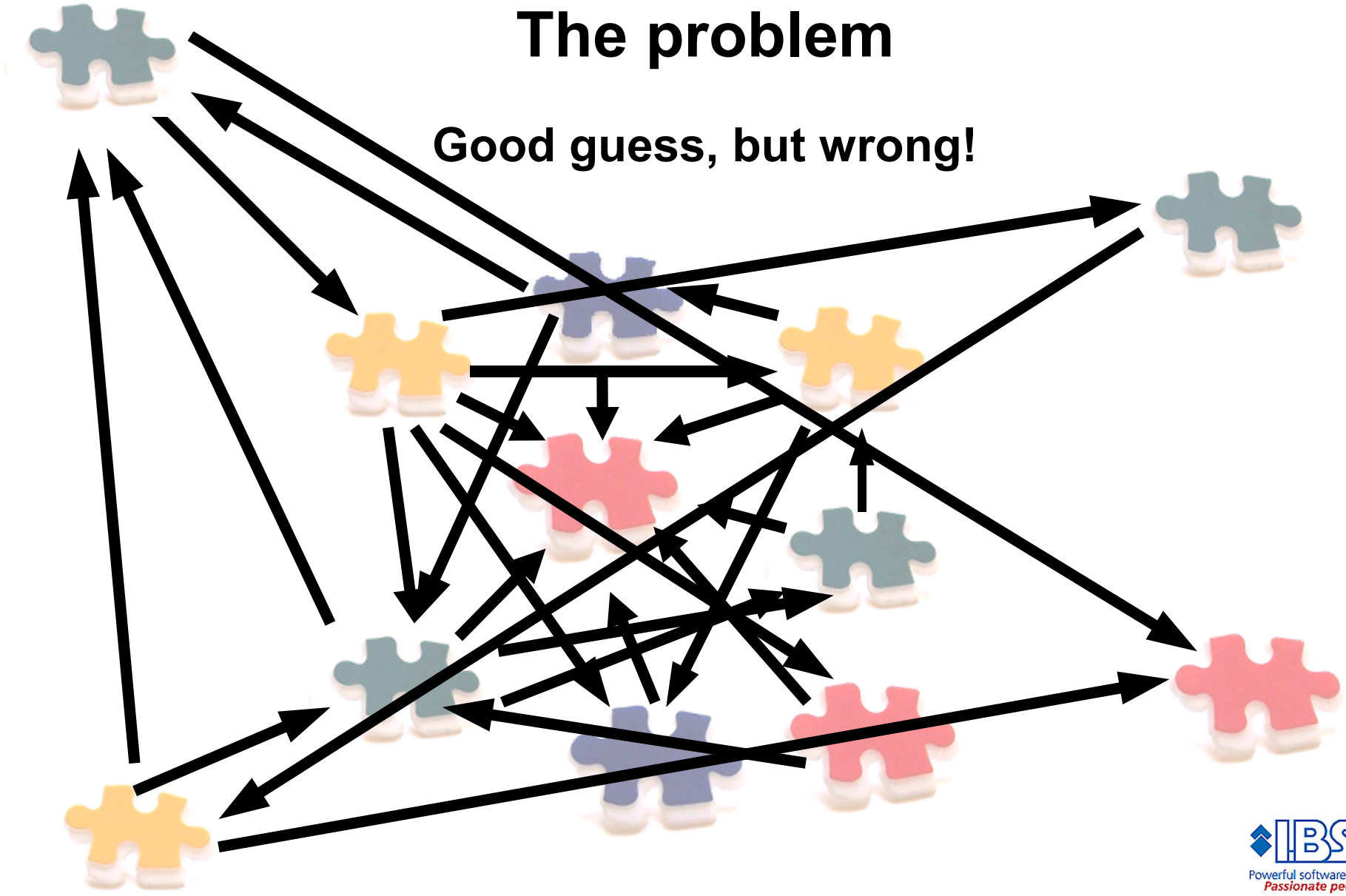
Version 1.2 – still works but but messy...

Contest: Guess what the next version looks like?



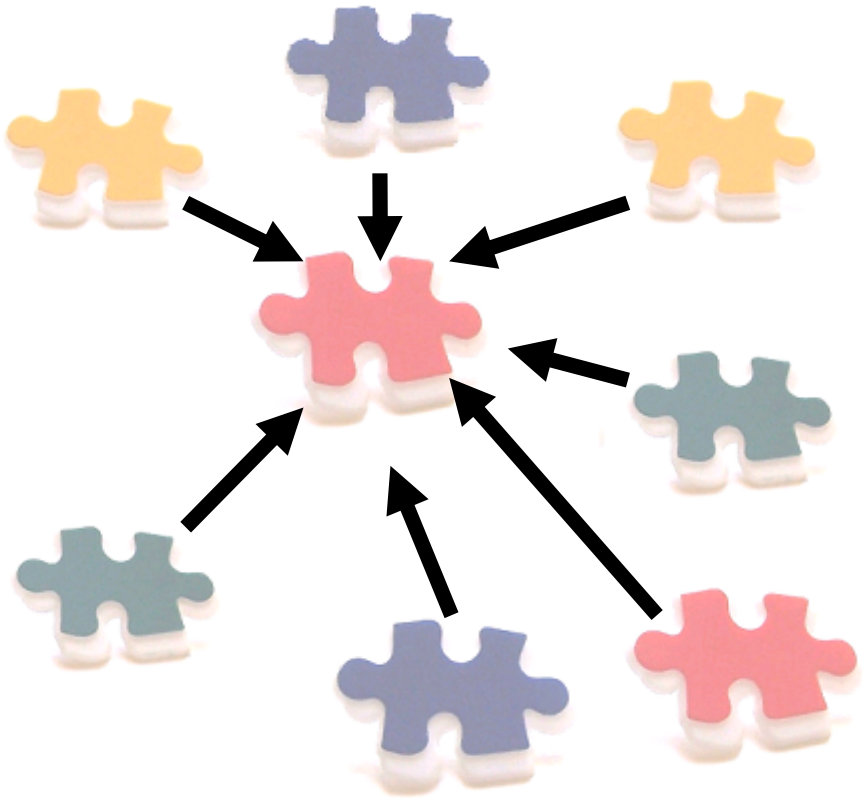
The problem

Good guess, but wrong!



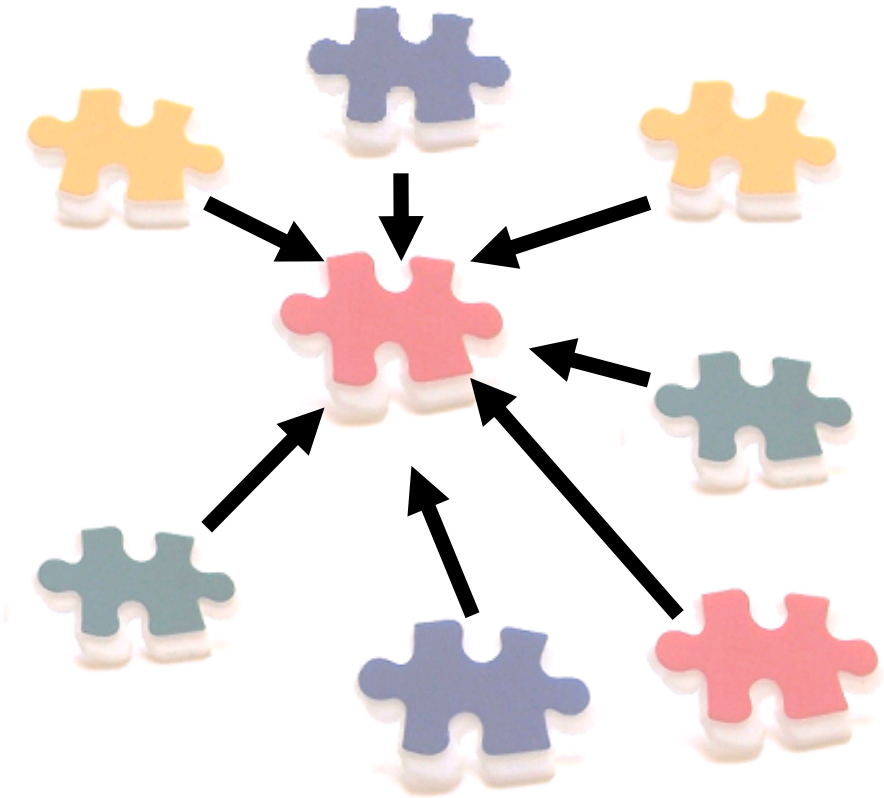
The problem

<u>What?</u> How is this possible?



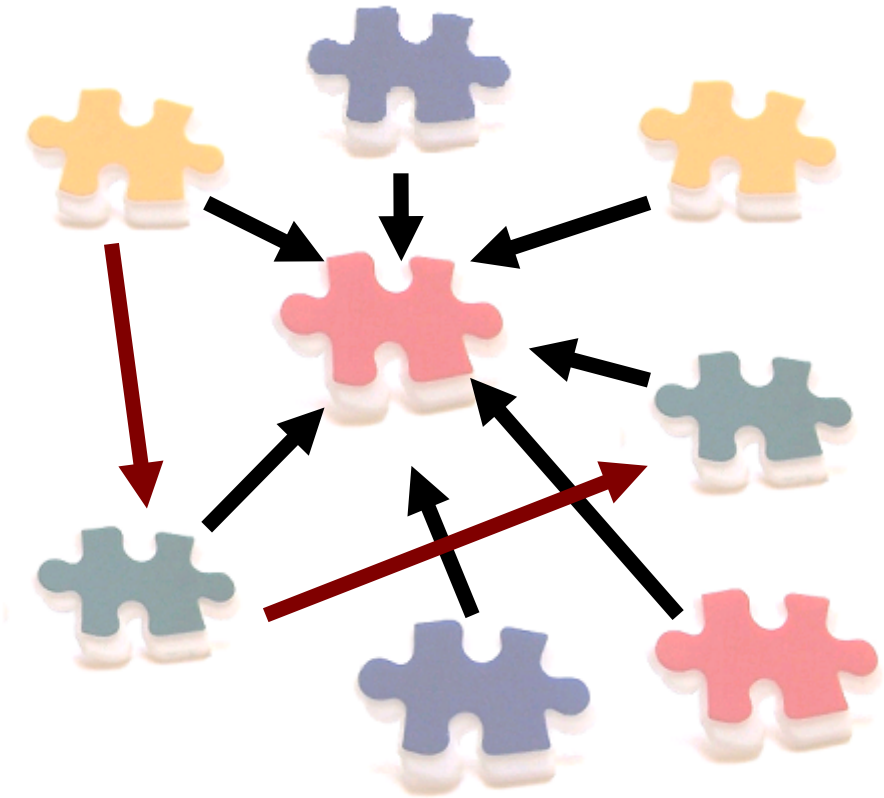
The problem

Version 2.0 – re-write, looking good again



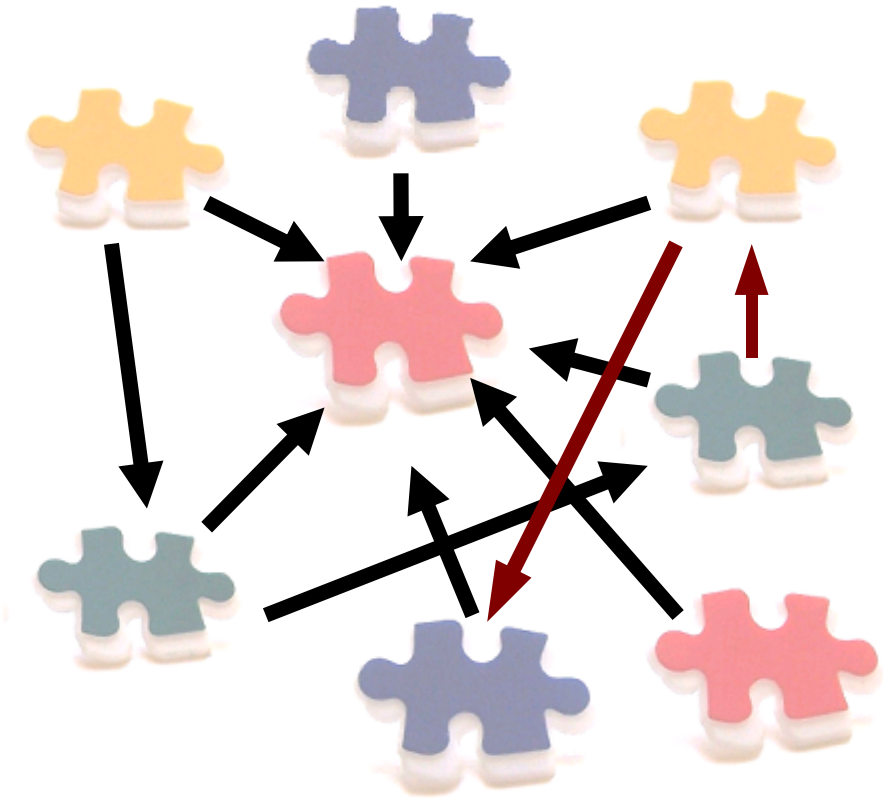
The problem

Version 2.1 – just needed a few hacks...



The problem

Version 2.2 – still works but but messy...



Other problems

- **Deployment**
  - We do not want to re-deploy the whole application, only the unit that was modified
  - Only the deployed unit is affected, no need to bring the whole system down
  - With a component based architecture this can be very clean and neat
  - The Java IDE:s does this a lot...
- **Release Management**
  - Possible to release features when ready, avoiding big bang releases (kind of agile)

The one and only historical slide

- **Component based architectures is nothing new**
  - Has been around for years
    - OSGi established in 1999
- **The interest picked up when Eclipse switched to OSGi**
- **All (or almost) application servers are going there**
  - BEA microService Architecture (OSGi)
  - JBoss Microcontainer (JMX based)
  - JonAS (OSGi)
  - WebSphere 6.1 (OSGi)
  - Apache Geronimo (OSGi)
  - GlassFish V3 (HK2)

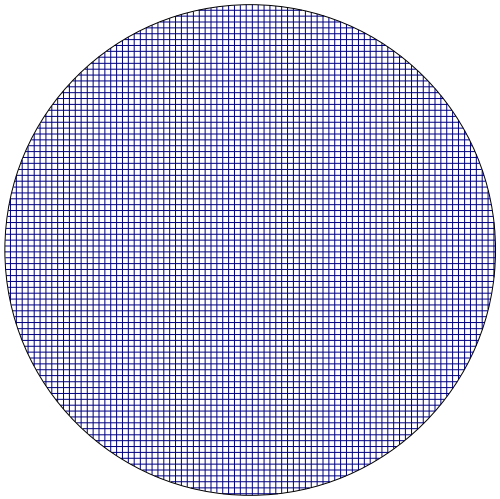
SD times about OSGi

”a quite contender for the title of most important technology of the decade”

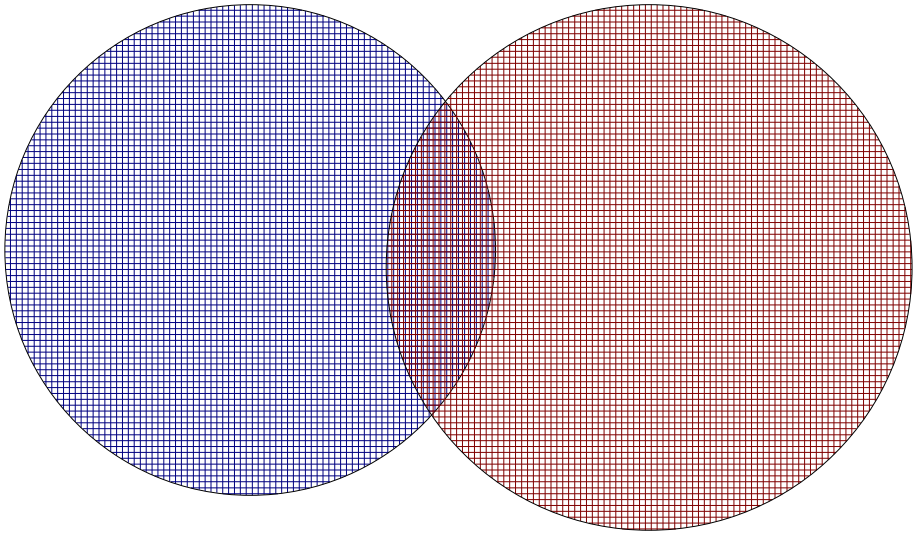
HK2 from 10K meters

- **Micro kernel for applications**
  - Module subsystem
  - Component/Service model
- **Applications are divided into modules**
- **Applications are executed in a Runtime container**
  - Supports Dependency Injection
  - Loads modules and resolves dependencies between modules
- **It is the foundation for GlassFish V3**

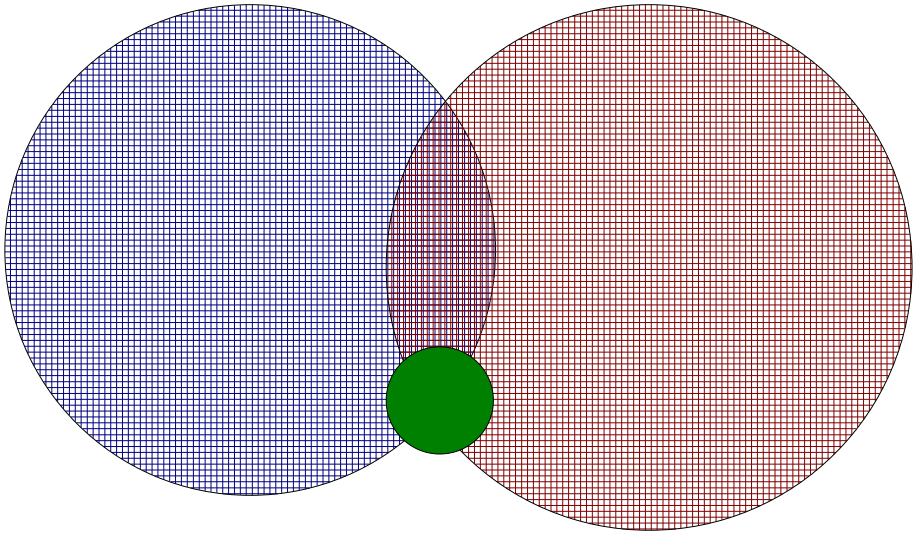
OSGi



OSGi + Spring



OSGi + Spring



HK2

DEMO

”What it is”

How HK2 is used in GlassFish V3

HK2 from 5K meters

- **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 (Service Provider Interface)**
- **Dynamic, components discovered at runtime**
- **Applications are "composed" in runtime**
- **Uses Dependency Injection (IoC)**

Why, how, and politics...

Why:

"HK2 does several things and it would exist even if we used OSGi. One of its roles is to encapsulate the dependency on the modular system so GFv3 could be easily switched from one system (JSR277) to another (OSGi)" - by Eduardo Pelegri-Llopart, Distinguished Engineer at Sun

How:

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

Politics:

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 Component Architecture (SCA)
  - Service Oriented Architecture (SOA)
  - Module / Component system

  SCA != SOA != Component system

- **A piece of code can have many names**
  - Service
  - Component
  - Module
  - Bundle
  - Plug-in

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 then ~5000 LOC (without comments)
  - Impressive design: clean and elegant
- **Runs on Java SE 5**
- **Current version is 0.2-SNAPSHOT**
- **License same as GFv3: GPLv2 and CDDL**

The two parts

- **The module subsystem**
  - The plug-in ecosystem
  - 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 Jar-files**
  - Meta information stored in the manifest
  - The manifest is created using a maven plug-in
    - Possible to do this manually
- **Declare their dependencies to other modules**
- **Has a life cycle**
  - Dynamically loaded and unloaded
- **Allows multiple version at the same time**
- **Module = plug-in = bundle = component**
- **Module != Service**

Dependencies

- Modularization will give you a better picture of what depend on what
- No need to look at the source code to find dependencies
- You can still end-up with “plug-in hell”, but at least you would know what the mess looks like

Module definition

- **A module is defined by**
  - Name, ”se.jsolutions.hk2.demo2”
  - Version number, ”1.2.3-rc1”
  - Imports (dependencies), ”se.jsolutions.hk2.demo1”
  - 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*
- **Possible to re-export an imported SPI**
  - Valuable for containers; GlassFish exports the Ruby container

Module definition: exports

- Defines the published (if any) API/SPI
- Classes that are not exported in a module can not be used by others

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**
- **Since applications are “composed” at runtime, they may break at runtime!**

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”
- **If you got a reference to a service, it can not go away**
  - unlike OSGi

Class Loading

- **To enforce the module contract only the public interface is visible to external user**
- **This is achieved with two class loaders:**
  - Public façade Class Loader
    - For classes in the SPI
  - Private Class Loader
    - For all other classes
- **The module subsystem can bootstrap itself**
- **No more classpath**
  - java -jar GlassFish.jar

Bootstrapping

- **A HK2 executable**
  - Does not have a main method/class
  - Is implemented as a set of modules
- **The bootstrap**
  - Not a HK2 module
  - Is loaded by the application class loader
  - Loads the module that implements the ModuleStartup interface
- **This allows the HK2 environment to be embedded**
  - The outer Java environment decides the class loader the bootstrap should be loaded with

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...
- **A maven based repository is handy**
  - You only need the bootstrap, the rest is fetched from the repository when needed

*** not implemented**

Modules Registry

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

I can do modularization anyway!

- **Yes, but:**

  "If the build and test environments do not enforce modularity, then the code is not modular"

Gregory Brail, John Wells, BEA Systems, "OSGi – The Good, The Bad, and the Ugly"

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**
- **State-full or state-less**
- **Declared by META-INF/services in the jar**
  - Generated by the maven plug-in
- **Two annotations**
  - @Contract
  - @Service

Annotations

Animal.java *

```
@Contract
public interface Animal {

    void bark();
}
```

6:1 INS

Annotations

Dog.java

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

Instantiation

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

```
@Inject(name=”Dog”)
Animal animal;

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

Dependency Injection in HK2

- **In HK2 dependency injection is clear and readable**

```
@Inject
SomeService service;
```

- **Reading the source code without the annotation it seams like the attribute is unassigned**
- **Be aware that using @Inject on setters (evil) are not so OO**

Services – the extension points

- **We want to avoid**

```
for (String arg : args) {
    if (arg.equals("ls")) executeLs(); else
    if (arg.equals("echo")) executeEcho(); else
    if (arg.equals("uname")) executeUname(); else
    showUsage();
}
```

Services

- **We can use Services**

```
@Contract
public interface Command {
    public void getName();
    public void execute();
}

@Service
public class Uname implements Command {
    ...
}
```

Services

- **And use dependency injection**

```
@Inject
Command[] commands;
...
for (String arg : args) {
    for (Command cmd : commands) {
        if (cmd.getName().equals(arg)) {
            cmd.execute(); break;
        }
    }
}
```

- **Possible to add options dynamically**

DEMO

”Extensibility”

Services

- **Decouples code**
  - Application code should be independent of the concrete implementation of the service interface
  - Isolates us from programming with modules directly
  - If every module would reference each other, all modules would be loaded at startup

Services example

```
public class Server {
   ...
   {
        // Thread that post new files to its handlers
        Scanner dir = new DirectoryScanner("/tmp");

        Handler pdf = new PDFHandler();
        dir.add(pdf); // Must be called before start()
        Handler png = new PNGHandler();
        dir.add(png); // Must be called before start()

        dir.start();
   }
```

Services example

Tight coupling

```
public class Server {
   ...
   {
        // Thread that post new files to its handlers
        Scanner dir = new DirectoryScanner("/tmp");

        Handler pdf = new PDFHandler();
        dir.add(pdf); // Must be called before start()
        Handler png = new PNGHandler();
        dir.add(png); // Must be called before start()

        dir.start();
   }
```

Services example

Tight coupling

```
public class Server {
    ...
    {
        // Thread that post new files to its handlers
        Scanner dir = new DirectoryScanner("/tmp");

        Handler pdf = new PDFHandler();
        dir.add(pdf); // Must be called before start()
        Handler png = new PNGHandler();
        dir.add(png); // Must be called before start()

        dir.start();
    }
```

Code grows

Services example

Tight coupling

```
public class Server {
```

Handler must know how to be used

```
    st new files to its handlers
    ew DirectoryScanner("/tmp");

    Handler pdf = new PDFHandler();
    dir.add(pdf); // Must be called before start()
    Handler png = new PNGHandler();
    dir.add(png); // Must be called before start()

    dir.start();
}
```

Code grows

Services example

```
public class Server {
    st new files to its handlers
    ew DirectoryScanner("/tmp");

    Handler pdf =
    dir.add(pdf  e start()
    Handler png =  );
    dir.add(png); // Must be called before start()

    dir.start();
}
```

Tight coupling

Handler must know how to be used

Static

Code grows

Refactor as Service

```
@Contract
public interface Handler {
}

@Service
public class PDFHandler implements Handler {
}

@Service
public class PNGHandler implements Handler {
}
```

Refactor as Service

```
// Thread that post new files to its handlers
DirectoryScanner dir = new DirectoryScanner();

@Inject
Handler[] handlers;

for (Handler handler : handlers) {
    dir.add(handler);
}

dir.start(); // Must be called after handlers are added
```

Scope

- **Services instances has a scope**
  - Singleton
  - Per thread
  - Per application
  - Or custom...
    - GridScope / RemoteScope
    - PooledScope
  - No “web scope” (request/session) out of the box
    - HK2 is not a web container
- **Scopes are Services themselves**
- **A scope is responsible for storing the service instance tied to itself**

DEMO

”Scope”

HK2 and OSGi differences

- **The intention with HK2 is not to replace OSGi**
- **HK2 and OSGi share many architecture concepts**
- **HK2 was developed as the architecture for GlassFish V3, not as a general purpose framework**
- **HK2 is more light weight, OSGi is a full blown framework**
- **HK2 is developed by the GlassFish community**
- **The OSGi specification is developed by the OSGi Alliance (more like JCP); fee $3,000 - $20,000 annually**
- **OSGi is well documented, mature, and well proven**
  - HK2 is “0.2-SNAPSHOT”
- **OSGi Compendium Services (R4)**
  - Log Service, Position, UpnP Service, and many more

HK2 and OSGi differences

- OSGi defines remote management
- HK2 is not designed for non-stop applications
  - Always expect RuntimeException when calling a service in OSGi
- Spring has added support for OSGi
- OSGi is kind of Class Loader on steroid
- OSGi is a specification with many implementations
- OSGi is supported
  - Training
  - Consultants

Transactions/Security/etc are missing...

- **Obviously by design – else it would not be an application micro kernel**
- **If you are building a an application using HK2 and need transactions you are free to choose**
  - JPA, Hibernate, etc
  - Spring

HK2 + Spring (*)

- Spring (2.5+) creates beans from META-INF/Services

(*) Has not been tested – should work in theory

HK2 pros and cons

- **Pros**
  - Small, well designed, and easy to grasp
  - Dynamic
  - Enforces modularization
  - Delivery can be more flexible
  - No more Jar-hell
  - Understandable injection
  - Less singletons
- **Cons**
  - Not for non-stop applications
  - Dynamic
  - No “real” singletons (this is a good thing)
  - Pooling, static is not static
  - May introduce plug-in hell (if you do things wrong)

When to use this architecture

- **Applications that need to be extended (plug-in based)**
- **Applications that are container based such as application servers**
- **Large applications that needs to start quickly**

- **The perfect application to utilize this architecture is:**

>> ANT <<

  - Would remove the classpath problems
  - Would be easier to add new tasks
  - LOC would be less
  - Repository of tasks (actually based on maven...)

When not to use this architecture

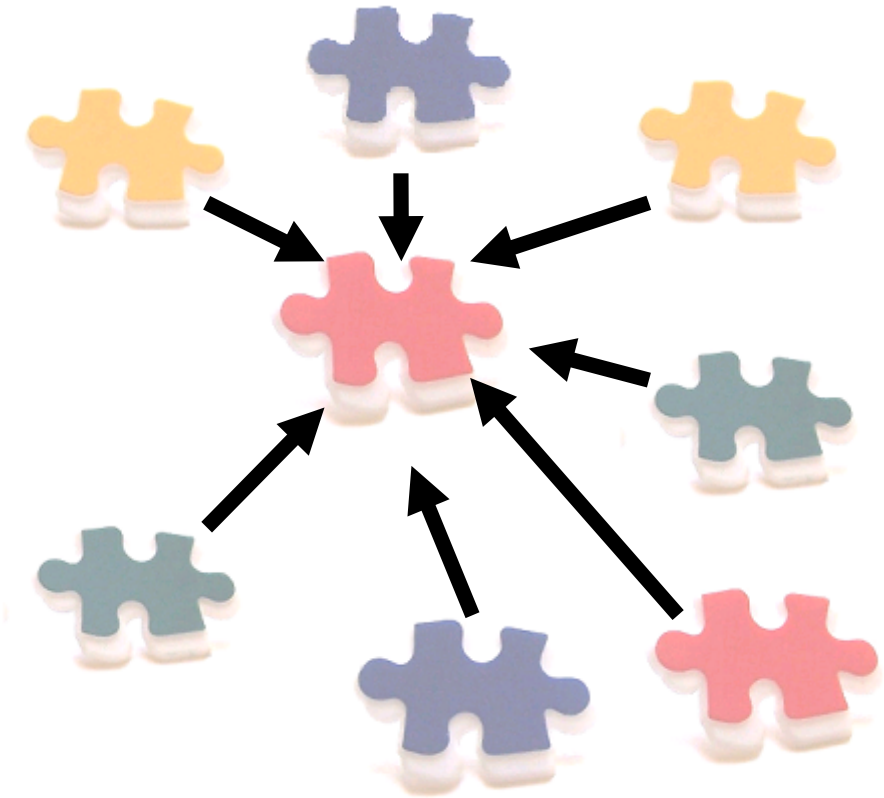
- **Static applications that have only one logical module**
  - The “jar” command
- **If you do not want Dependency Injection**
  - (maybe) If you only have one implementation of a Service

Summary

- **Component based architectures are here to stay**
- **Forecast**
  - More and more software will be developed with this kind of architectural concepts
    - Spring is going there
    - All application servers are going there
    - Java SE 7 is going there
    - Our guess it that Java EE 6 or 7 will go there
  - There will be business knowing this kind of architectures

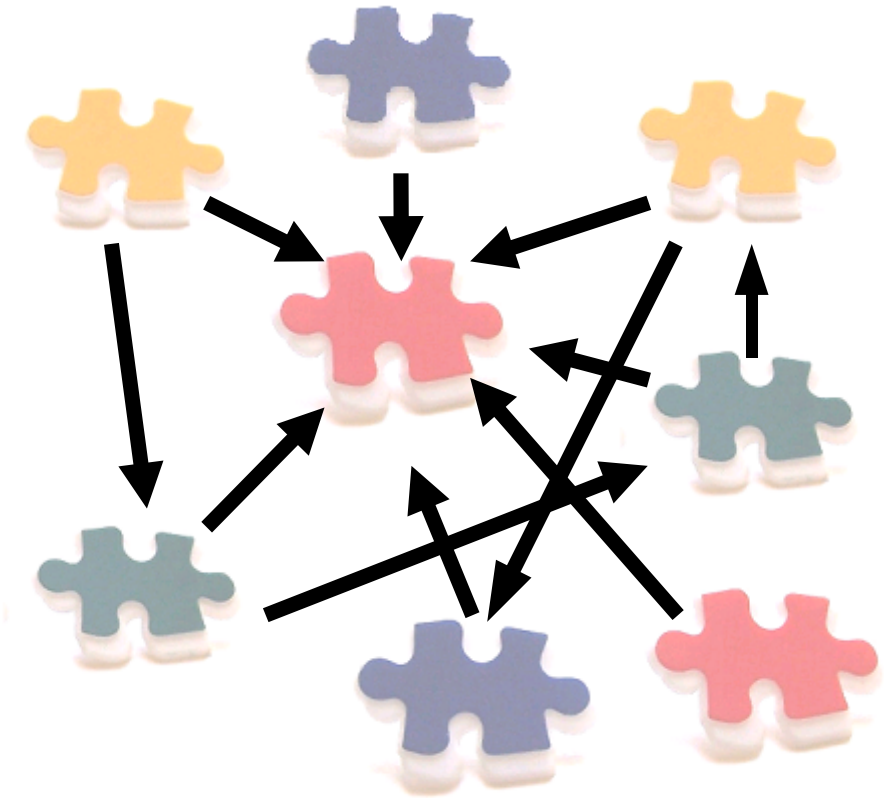
Summary

Makes it possible to keep a good structure



Not a silver bullet

So you might still end up in a mess if done incorrect



Other component / module subsystem technologies

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

(the not so) Famous last words...

- **It is a common misconception that using the same plug-in system would make it possible to mix and match**
- **It would <u>not</u> be possible to use Eclipse plug-ins in Netbeans if Netbeans used the OSGi framework**
  - The Eclipse platform is not the same as the Netbeans platform
  - If they where, they would be the same application

Developing a HK2 module with NetBeans 6.0

IBS
Powerful software
Passionate people

References

- **Presentation and source will be available at**
  http://jsolutions.se
- **HK2 web site**
  https://hk2.dev.java.net/
- **Glassfish v3 Engineers Guide**
  http://wiki.glassfish.java.net/Wiki.jsp?page=V3EngineersGuide
- **JSR 277: Java Module System**
  http://jcp.org/en/jsr/detail?id=277
- **OSGi™ - The Dynamic Module System for Java™**
  http://osgi.org/
- **Wikipedia about Dependency Injection**
  http://en.wikipedia.org/wiki/Dependency_injection

Q & A

About the authors

- **Rikard and Ferid is 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 one of the founders as well as a board member of the Swedish Java User Group “Javaforum Sweden”. 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 Sweden”. Ferid holds a B.Sc degree in Informatics.

Developing a HK2 module with NetBeans 6.0

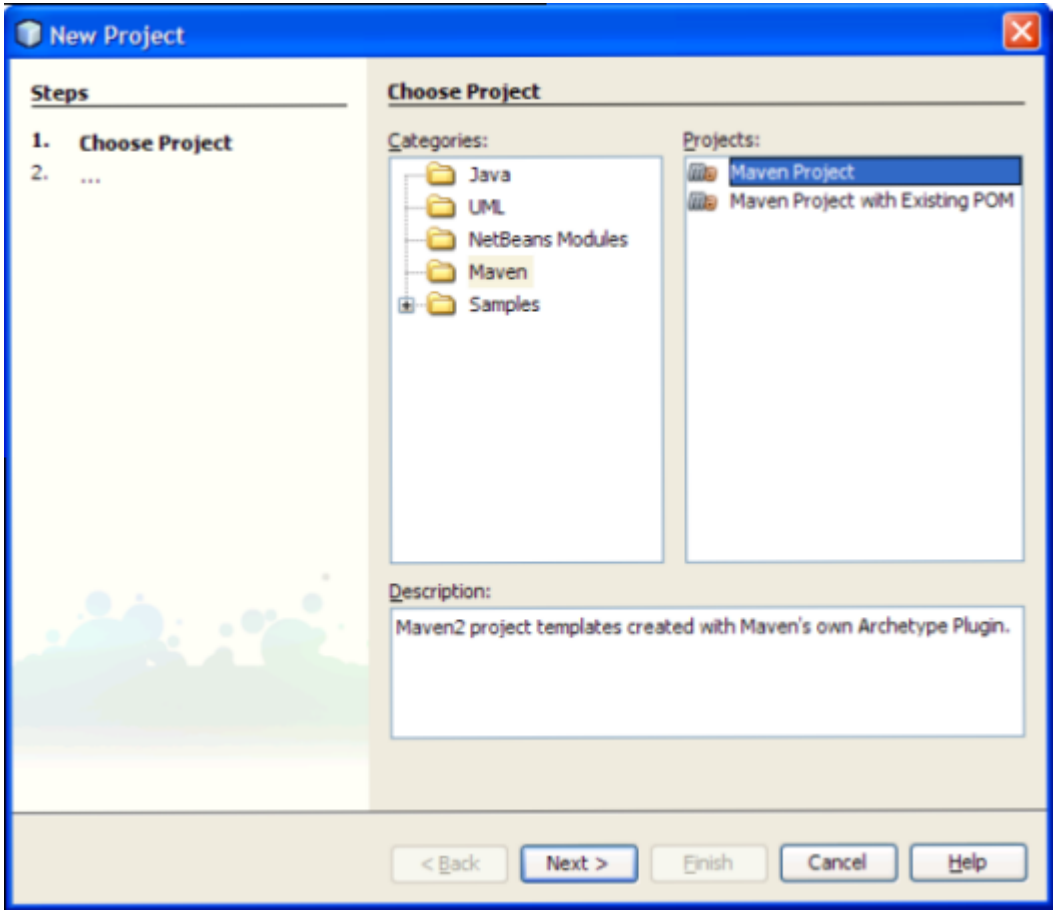
IBS
Powerful software
Passionate people

Developing a HK2 module with NetBeans 6.0

- This tutorial will show you how to develop the famous “Hello World” as a HK2 module using NetBeans 6.0/maven

Developing a HK2 module with NetBeans 6.0

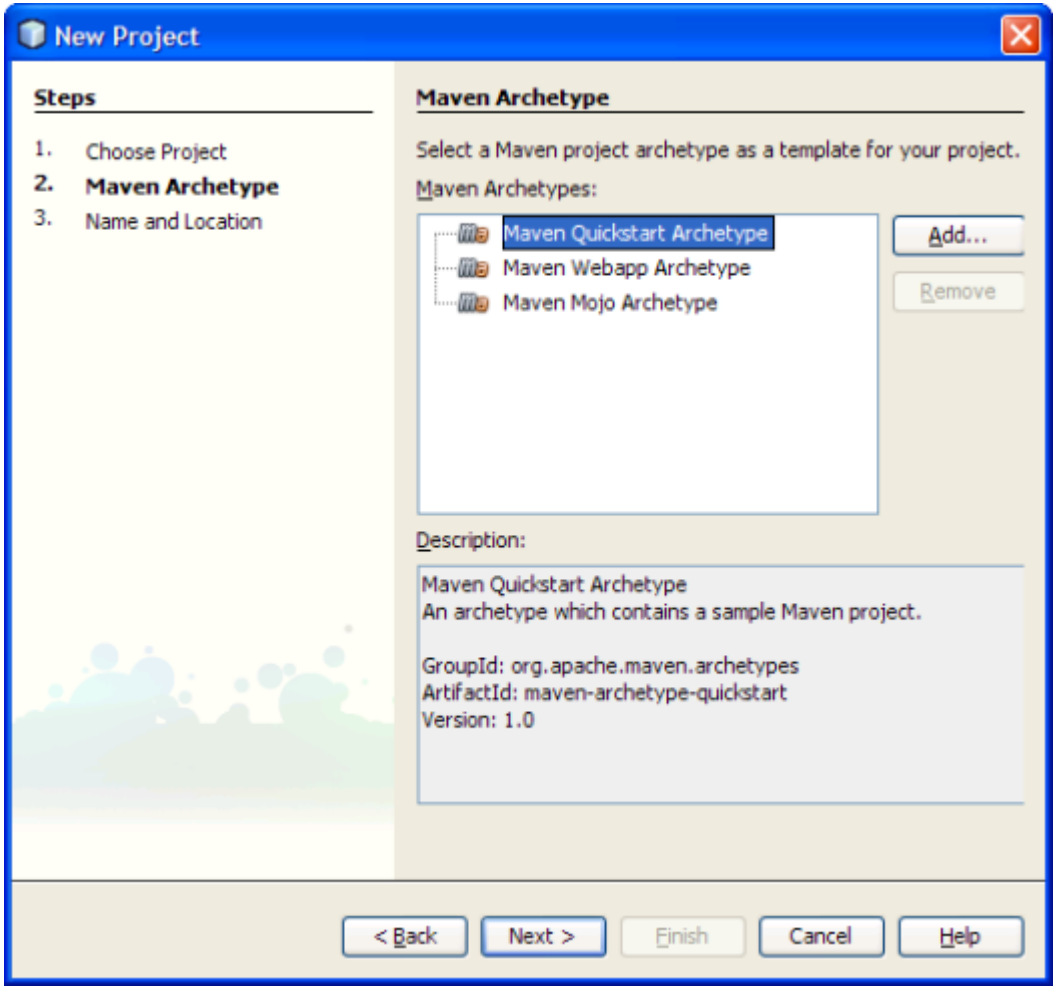
- **The very first step is to create a new Maven Project**



Powerful software
Passionate people

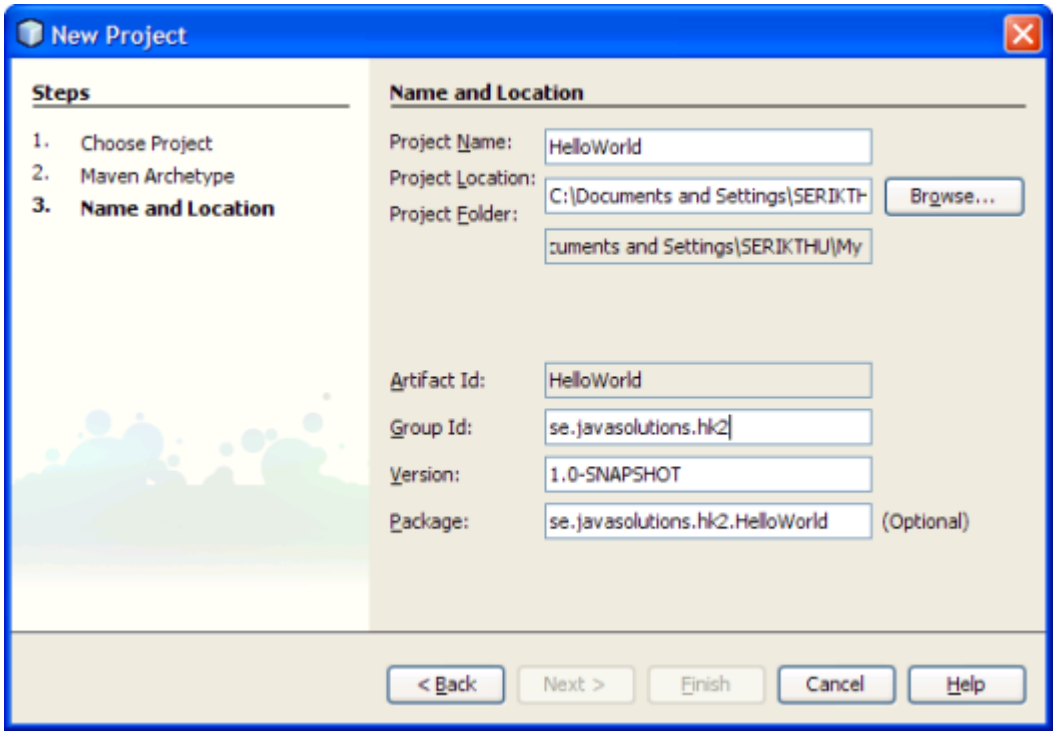
Developing a HK2 module with NetBeans 6.0

- **Maven Quickstart Archetyp:**



Developing a HK2 module with NetBeans 6.0

- We also must supply the artifact id, group id and the version



- NetBeans actually creates a Hello world source file named App in the package se.javasolutions.hk2.HelloWorld

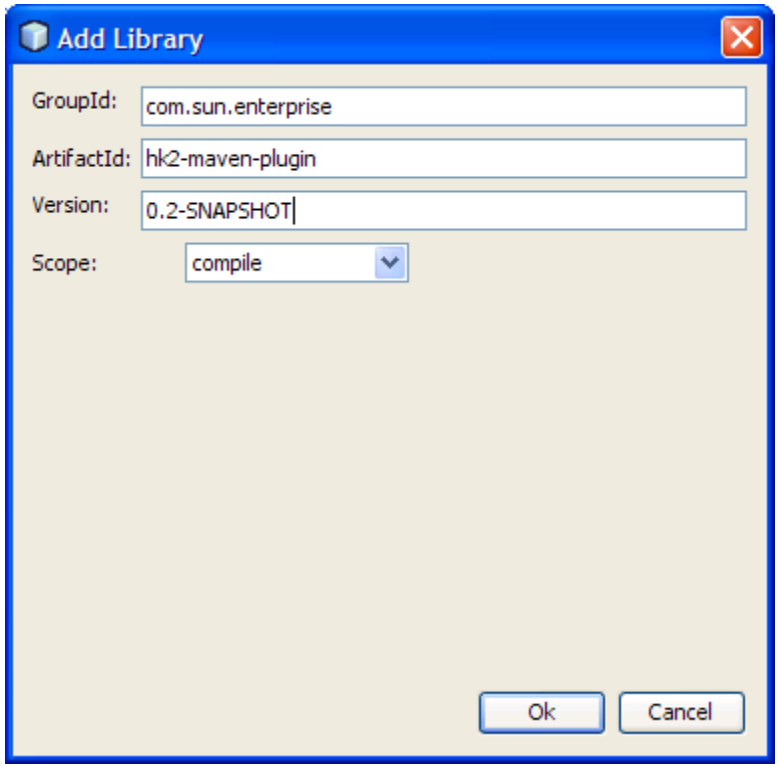
Developing a HK2 module with NetBeans 6.0

- **The next thing we need to do is to add the maven repository containing HK2. The pom is located in the Project Files folder. Add the following to pom.xml**

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>se.javasolutions</groupId>
    <artifactId>helloworld</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>helloworld</name>
    <url>http://maven.apache.org</url>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
    <repositories>
        <repository>
            <id>gfv3</id>
            <url>http://download.java.net/maven/glassfish/</url>
        </repository>
    </repositories>
</project>
```

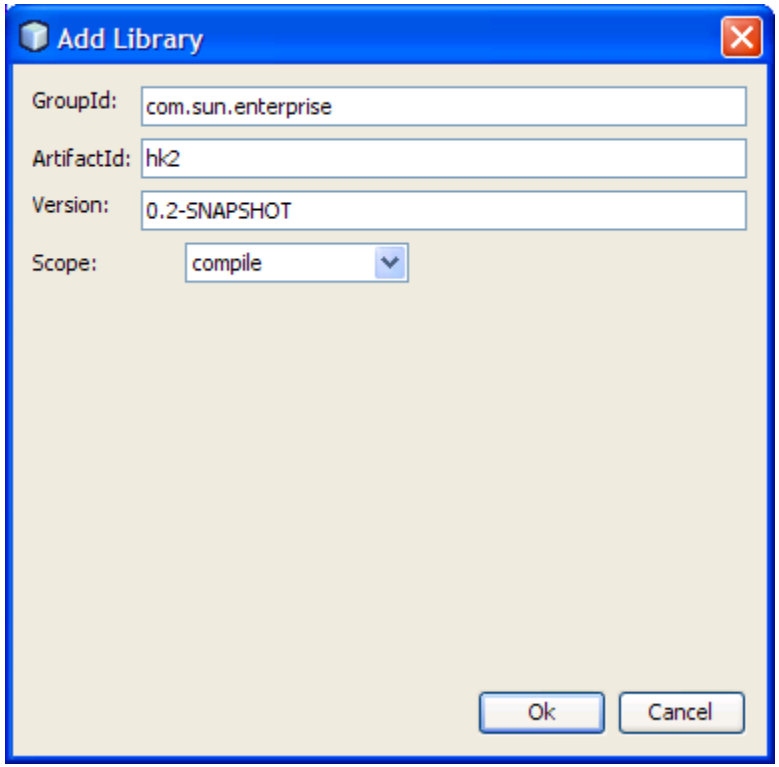
Developing a HK2 module with NetBeans 6.0

- Next we need to add the dependency to the hk2-maven-plugin. To do this, select Add Library on the Libraries folder in the project. Add the following



Developing a HK2 module with NetBeans 6.0

- We also need to add a dependency to hk2 iteself



Developing a HK2 module with NetBeans 6.0

- **Now we must change the pom file to use the hk2-maven plugin to build and package the the project**

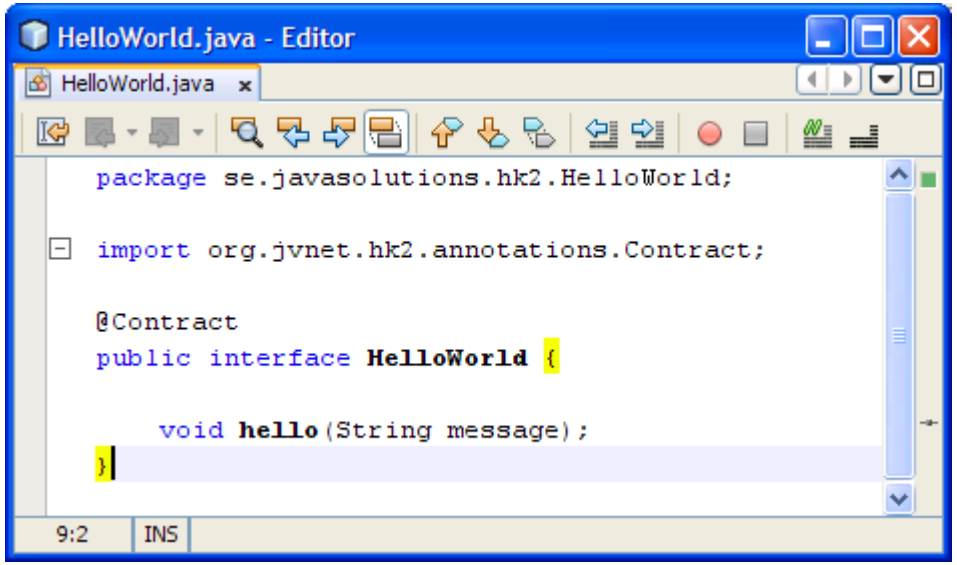
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www
    <modelVersion>4.0.0</modelVersion>
    <groupId>se.javasolutions</groupId>
    <artifactId>helloworld</artifactId>
    <packaging>hk2-jar</packaging>
    <!--    <packaging>jar</packaging>-->
    <version>1.0-SNAPSHOT</version>
    <name>helloworld</name>
    <url>http://maven.apache.org</url>
    <build>
        <plugins>
            <plugin>
                <groupId>com.sun.enterprise</groupId>
                <artifactId>hk2-maven-plugin</artifactId>
                <version>0.2-SNAPSHOT</version>
                <extensions>true</extensions>
            </plugin>
        </plugins>
    </build>
    <dependencies>
```

Developing a HK2 module with NetBeans 6.0

- **By default NetBeans assumes that we are using Java 1.4 and will therefor not recognize annotations**
  - Select Properties on the project folder and the category source
  - Change Source/Binary Format to 1.5/1.6.
  - Close and re-open the project

Developing a HK2 module with NetBeans 6.0

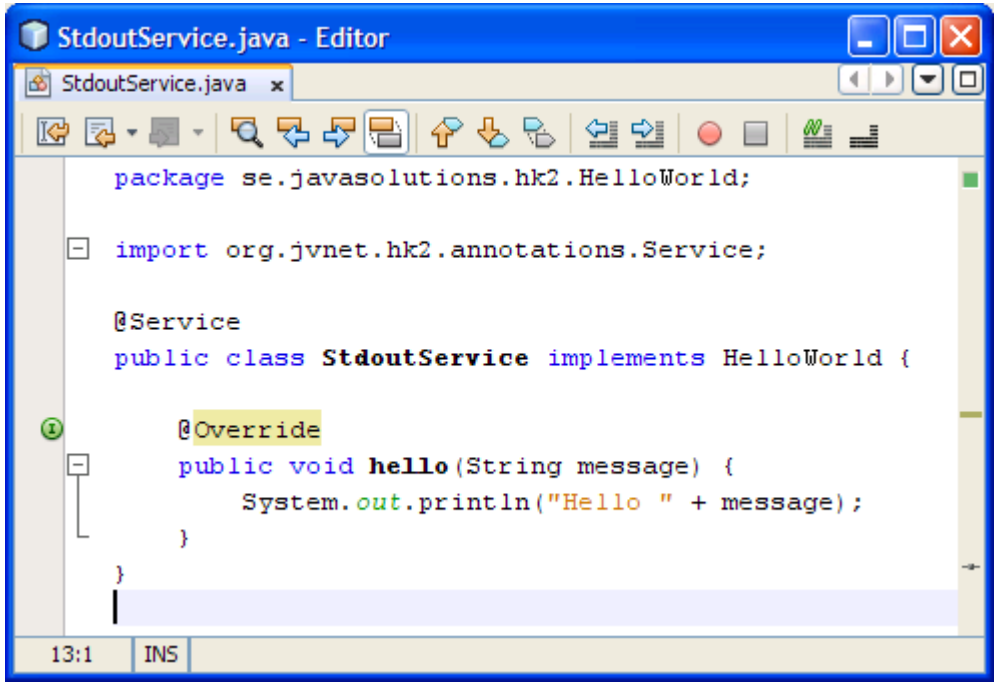
- **Now it is time to create the contract for our hello world service**



- **As you can see is an ordinary interface with an @Contract annotation.**

Developing a HK2 module with NetBeans 6.0

- **The Service implementation is just as simple:**



Developing a HK2 module with NetBeans 6.0

- **The App class is modified to implement the interface ModuleStartup**

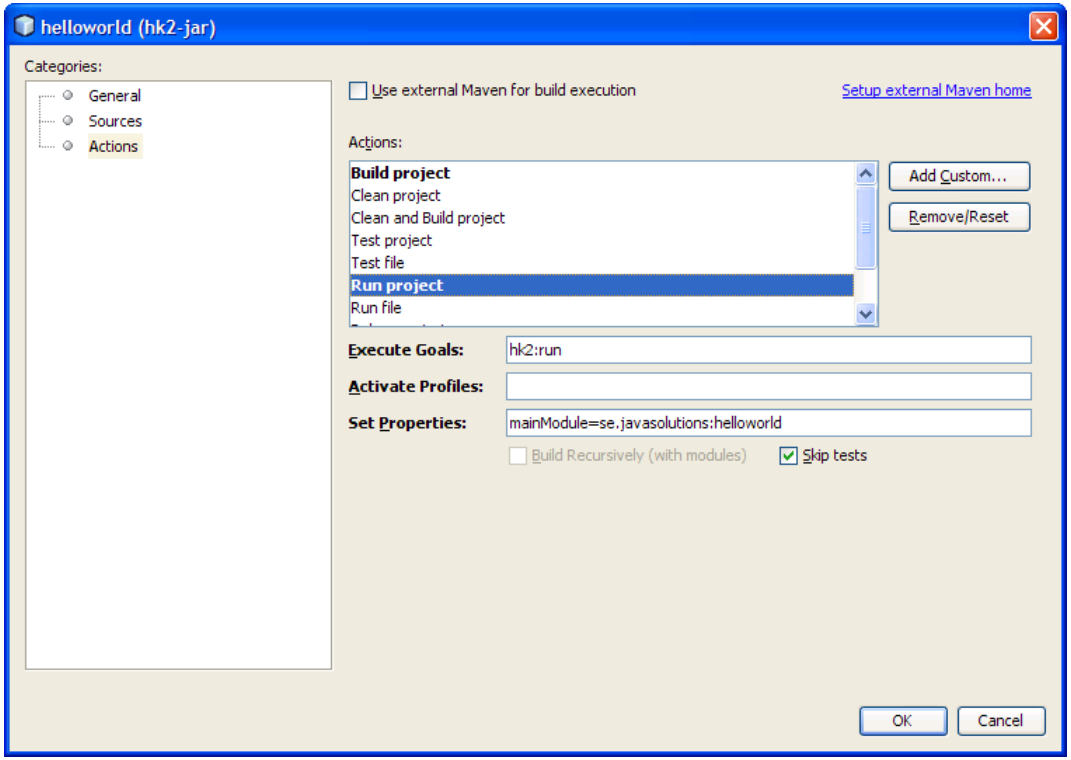
App.java - Editor

```
package se.javasolutions.hk2.helloworld;

import com.sun.enterprise.module.bootstrap.ModuleStartup;
import com.sun.enterprise.module.bootstrap.StartupContext;
import org.jvnet.hk2.annotations.Inject;
import org.jvnet.hk2.annotations.Service;
import se.javasolutions.hk2.HelloWorld.HelloWorld;

@Service
public class App implements ModuleStartup {

    @Inject
    HelloWorld service;

    public void setStartupContext(StartupContext ctx) {
    }

    @Override
    public void run() {
        service.hello("foobar");
    }

}
```

13:24 INS

Developing a HK2 module with NetBeans 6.0

- The service instance is injected to the service field by the HK2 runtime container (previous picture)
- And the very final thing we must to is to modify run project action to (note the Set properties)



Developing a HK2 module with NetBeans 6.0

- **Now you can run the project**
- **The complete NetBeans project with sources can be downloaded from:**
  - http://jsolutions.se/wp-content/uploads/2008/01/helloworld.zip
- **Feel free to send commends to rikard.thulin(at)ibs.se**

- **References**
  - HK2 development site
    http://hk2.dev.java.net/
  - GlassFish v3 Engineering Guide
    http://wiki.glassfish.java.net/Wiki.jsp?page=V3EngineersGuide