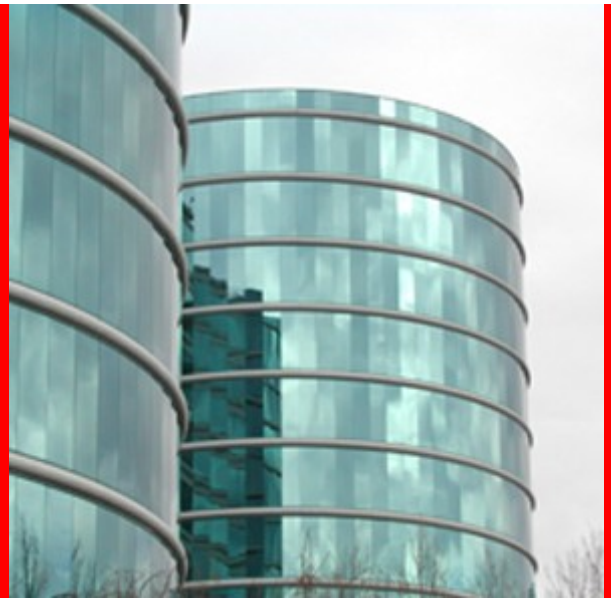


ORACLE®

HK2 and configuration

Jerome Dochez
GlassFish architect

HK2

- Services for server side Java
- HK2 = Hundred Kilobytes Kernel = HKK = HK^2
- Started as a full fledge module runtime
- Scoped reduced after OSGi adoption :
  - Module management
  - OSGi API isolation
  - Service based runtime provider
  - Component definition
  - Configuration handling
- Separate project from GlassFish

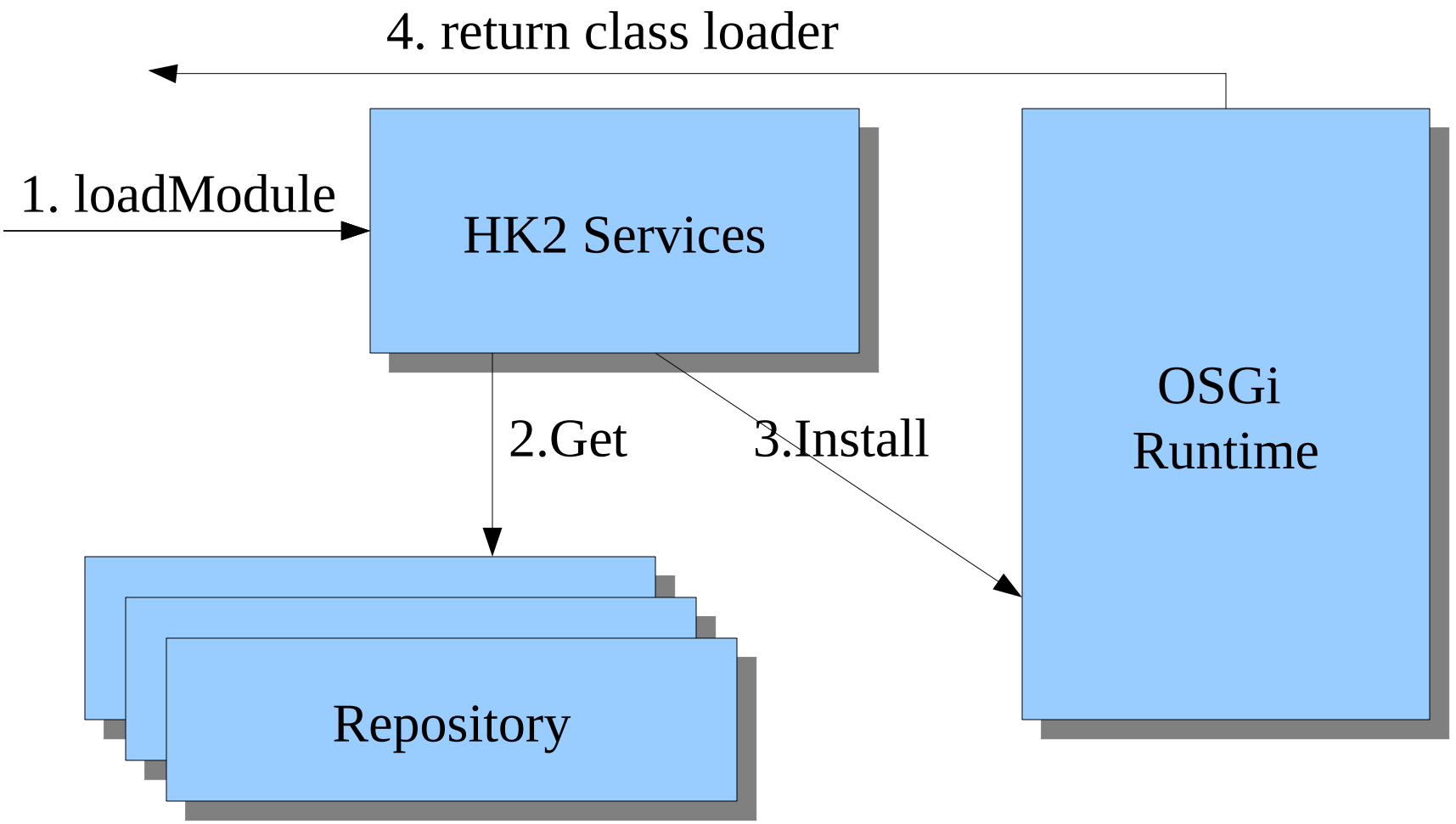
Agenda

- Module Management and OSGi
- Startup/shutdown
- Hk2 Components
- Configuration
  - Definition
  - Extensibility
  - REST
- Admin commands framework
- Deployment framework

Module Management

- Module Repository
  - Directory
  - Maven repository
- Lazy startup of modules based on service lookup
- Use OSGi runtime underneath
- Service Lookup resolve to module, module gets loaded in OSGi

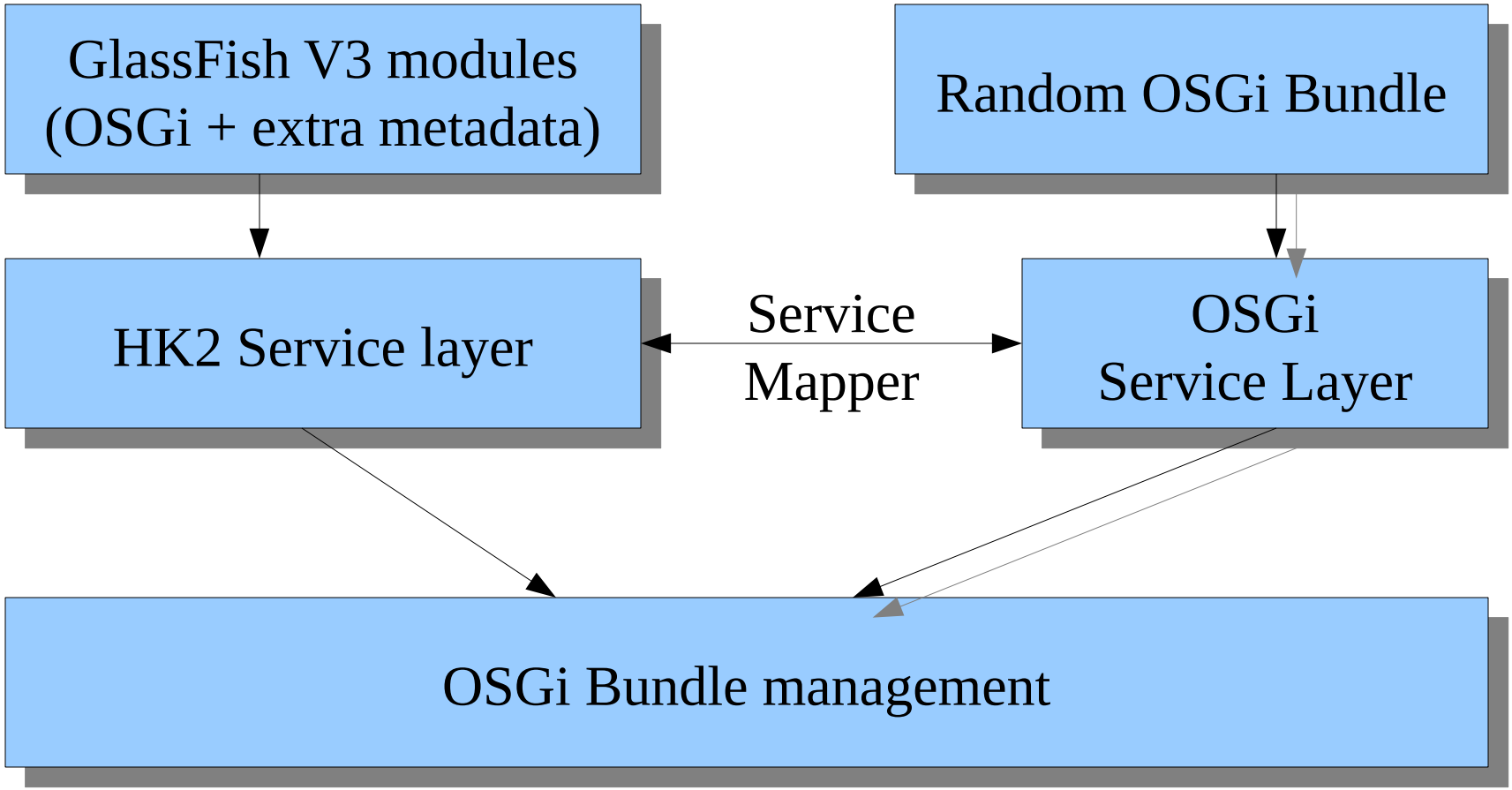
GlassFish V3 Module management



OSGi API isolation

- Started from 277-like implementation
- When running embedded, GlassFish modules are all collapsed into one uber-jar.
  - No dynamic feature
  - Basic module management (remember only one jar).
  - Super fast startup (700 ms)
- Easy to provide different implementations for different OSGi runtime (not necessary up to now).
- OSGi is complicated, better to hide it to mainstream developers in GlassFish

GlassFish V3 Runtime



Startup/Shutdown sequences

- Startup and shutdown use special contracts
- Based on run-level or priority
- Multi-threaded
- Some startup services are optionals, others failure to start will provoke an immediate server shutdown.
- Different distributions can have a different set of services (hence startup/shutdown sequences).

Service Based Runtime

- Services are declared with @Contract Annotation

```
@Contract
public interface Wombat {
    void someContractMethod(String param);
}
```

- Tag interface that denotes a startup service

Service Based Runtime

- Contracts implementations are annotated with @Service
  - Optional name
  - Scope (singleton, perLookup, threaded)
  - No scope annotation = singleton

```
@Service
Public class AustralianWombat implements Wombat {
  void someContractMethod(String param) {
    …
  }
}
```

Service Lookup

- Habitat contains all the known services in GlassFish v3.
  - habitat.getAllByContract(Startup.class);
    will return List<Startup>
- V3 startup code (simplified)

```
for (Startup startup : habitat.getAllByContract(Startup.class)) {
    Logger.info("Started " + startup);
}
```

Component

- Services are components, they have a scope (lifecycle), the have dependencies and provide contracts implementation
- Dependencies of a service are expressed with
  - @Inject
- Injected resources can be resources themselves
- Service allocation cascading based on dependencies resolution.
- Injected resources are looked up from the Habitat

Server Initialization

- On startup, modules are introspected to determine :
  - List of available contracts (indexes)
  - List of services implementing such contracts.
- Lazy lookup of services is performed, even no class-loader is created until a service is actually requested (use of a manifest style service information file called the inhabitant file).
- Experimenting with ASM to be able to use normal jar rather than enforcing hk2-jar (presence of the inhabitant file is mandatory so far).

Component Lifecycle

- PostContruct
  - Interface implemented to have a postConstruct method called after injection is performed.
  - Constructor cannot be intercepted.
  - After postConstruct is called, service is installed in the habitat.
- PreDestroy
  - Interface called when service is removed from the habitat.
  - Hook for cleanup.

Tricks to remember

- Nobody calls services any more, they call contracts implementation.
- Services are dynamic, remove a optional jar from the modules directory and its services are not available.
- Services use injection to get initialized, very little lookup exists in v3.
- Services initialization will result in multiple sub services cascade initialization :
  - No multi-threading
  - No circular references support

Configuration

- Special type of components to read/write configuration data to the domain.xml
- Mapping is defined via interfaces (kind of like JAXB)
- Interfaces are annotated with @Configured annotation
  - Fields are annotated with @Attribute
  - Sub-elements are annotated with @Element
- Supports subclassing and extensibility

Configuration Example

```
@Configured
public interface Server extends … {

        @Attribute(key=true)
        String name();

        @Attribute String description();

        @Element Replication replication;
}
@Configured public interface Replication extends....
```

Configuration extensibility

```
@Configured
public interface Server extends … {
        @Element(“*”)
        public List<Module> modules
}
@Configured
public interface RailsModule extends Module {
        @Attribute String name();
}
@Configured
public interface WebModule extends Module {...}
```

Configuration Extensibility (2)

```
<server>
        <rails-module name=”foo”/>
        <rails-module name=”bar”/>
        <web-module name=”xyz”/>
</server
```

Existing extensible hooks

- Container to add container specific configuration
  - In Config.java :
    - @Element("*")
    - List<Container> getContainers();
- Application to add per application per container configuration
  - In Engine.java
    - @Element("*")
    - List<ApplicationConfig> getApplicationConfigs();
- Monitoring, to add application/container monitoring configuration (tbd)

Configuration implementation

- Based on streaming parser
- One class implements all @Configured interface (Dom.java)
- Each instance of Dom creates a proxy view of the data, the proxy type is the @Configured annotated type.
- User's code use interface based access, all configuration data is stored in a network of Dom instances.

Views

- Configuration data has different “Views” attached to dom instances
  - Default view ⟶ TranslatedView
    all ${propertyname} properties are resolved, user's get the translated values.
  - RawView
    all ${propertyname} properties are not resolved, user's will see the domain.xml raw values like ${propertname}
  - WriteableView
    Used to mutate a configuration objects (always within a Transaction semantics).

@DuckTyped

- Ability to add methods implementation to an interface.
- To the interface implementation, the methods is already implemented.
- Users invoke such methods like any other interface methods.
- Works because @Configured interfaces are managed objects so we can intercept all calls and redirect them

Transactions

- Simple transactions must be started to change configuration objects.
- Mutated object must be identified to the transaction.
- Concurrent transactions can happen as long as they don't mutate the same configuration instances.
- Transaction are either committed or rollbacked.
- Committed transaction are written to the domain.xml and listeners are notified

Configuration Listener

- Code that wish to be notified when configured instances are mutated must implement a ConfigListener interface
- 1 method : changed(PropertyChangeEvent[] events)
- Listeners can consume a change or deny it
  - Denied changes involves a server restart.
  - Listener cannot revert changes, changes have been applied and they are just notified after the facts.

Configuration Upgrade

- ConfigurationUpgrade is a contract
- Upgrades are ran on startup, very early
  - Before Init and Startup services are executed.
  - V3 code never has to deal with old configuration, by the time V3 is executed, the domain.xml has been upgraded.
- Asadmin start-domain –upgrade to run the upgrade and exit the application server
- @DuckTyped methods can help API backward compatibility.

Configuration Additional services

- Mbeans
  - All configured interfaces instances can be automatically registered in the mbean server.
- Generic admin commands
  - CRUD style commands can be provided by the framework allowing to change, create or delete configuration entries with no extra code.
- REST
  - All configured interfaces instances are automatically accessible through our REST interface

REST Interface

- Domain level configuration available at :
  - **http://localhost:4848/management/domain**
- Binding automatic for all @configured interface
- @RestRedirect when rest commands are expected to trigger a command invocation rather than just changing the configuration (e.g. deploy).
-

Rest Redirect example

```
@Configured
@RestRedirects(
      {
       @RestRedirect(opType= RestRedirect.OpType.DELETE,
commandName="undeploy"),
       @RestRedirect(opType= RestRedirect.OpType.POST,
commandName = "redeploy")
      }
)
public interface Application extends Injectable,
ApplicationName, PropertyBag {...}
```

Admin command framework

- CLI framework:
  - Command invocation engine
  - Parameter parsing
  - Remote invocation and result display
- Server Side framework
  - Leverage HK2 components
  - Contract : AdminCommand
  - One Method :
    - ActionResult execute(Context context).
  - Parameters are injected, using the @Param annotation

More information

- hk2.dev.java.net

  For the motivated readers :
  - OSGi R4

- Similar technologies
  - Xbeans, web-beans, spring all have used component definition, injections and other similar techniques to help developer's productivity

Admin Command Example

```
@Service(name = "create-http")
@Scoped(PerLookup.class)
@I18n("create.http")
public class CreateHttp implements AdminCommand {

    @Param(name = "protocolname", primary = true)
    String protocolName;

    @Param(name = "request-timeout-seconds", defaultValue = "30",
optional = true)
    String requestTimeoutSeconds;

    @Inject
    Configs configs;
```

Deployment framework

- Java EE agnostic
- Handles any type of container and jar format
- Comes with :
  - Admin commands to deploy/undeploy
  - Configuration entries to be added in central configuration
  - Pluggable interfaces for containers
- See my blog : wombat container
  - http://blogs.sun.com/dochez

What's next

- CDI in place of HK2 components
- Removal of hk2-jar (plain old jar file support)
- Moving reusable pieces from v3 code base into hk2.
- Better Mbean support (if customer demand)
- Multi-threaded components initialization (when cascading).
- Requirements from WLS and CEP