

Kraken

Introduction to the
OSGi based security platform

xeraph@nchovy.com

○ Target Audience

- Information security solution developer
- Java developer who is interest on OSGi application stack

○ Security Platform

- Most security solutions shares common requirements
 - Aggregate some informations from agents or sensors
 - System status (performance, software versions, etc)
 - Log from various data store (file, database, syslog, snmp trap, etc)
 - Inspect informations and applies security policies
 - Trigger configured actions
 - Send event or alarm through various media (e.g. sms, email)
 - Control other security solutions (e.g. firewall policy)
 - Drop packet
 - Generate statistics and reports
- Security Platform provides
 - common functionalities as components
 - reliable and solid development framework

○ Common Problems

– Lack of pipeline architecture style

• Security solutions need pipelined message processing

– IDS message flow

- » Generate intrusion event
- » Apply response policy to event
- » Respond to an intrusion (syslog, trap, sms, RST packet, etc)

– ESM message flow

- » Read logs from file, database, network, and so on.
- » Parse logs and normalize them
- » Filter logs by policy
- » Send event to realtime view of console
- » Summarize logs and generate reports

○ Common Problems

- Lack of pipeline architecture style
 - Solutions that did not consider pipelining encounter following problems:
 - Hard wiring of message path
 - » You should modify caller side if you want to add new feature
 - Hard to provide optional feature and remove unused feature
 - » You should maintain many branches
 - Long downtime
 - » You should kill the process, patch some files, and start it
- Lack of troubleshooting support
 - Do you think logging is sufficient?
 - Error log does not provide detail information
 - It's not that easy to trace a tons of debug logs
 - Add debug log to code and wait until problem appears again?

○ Common Problems

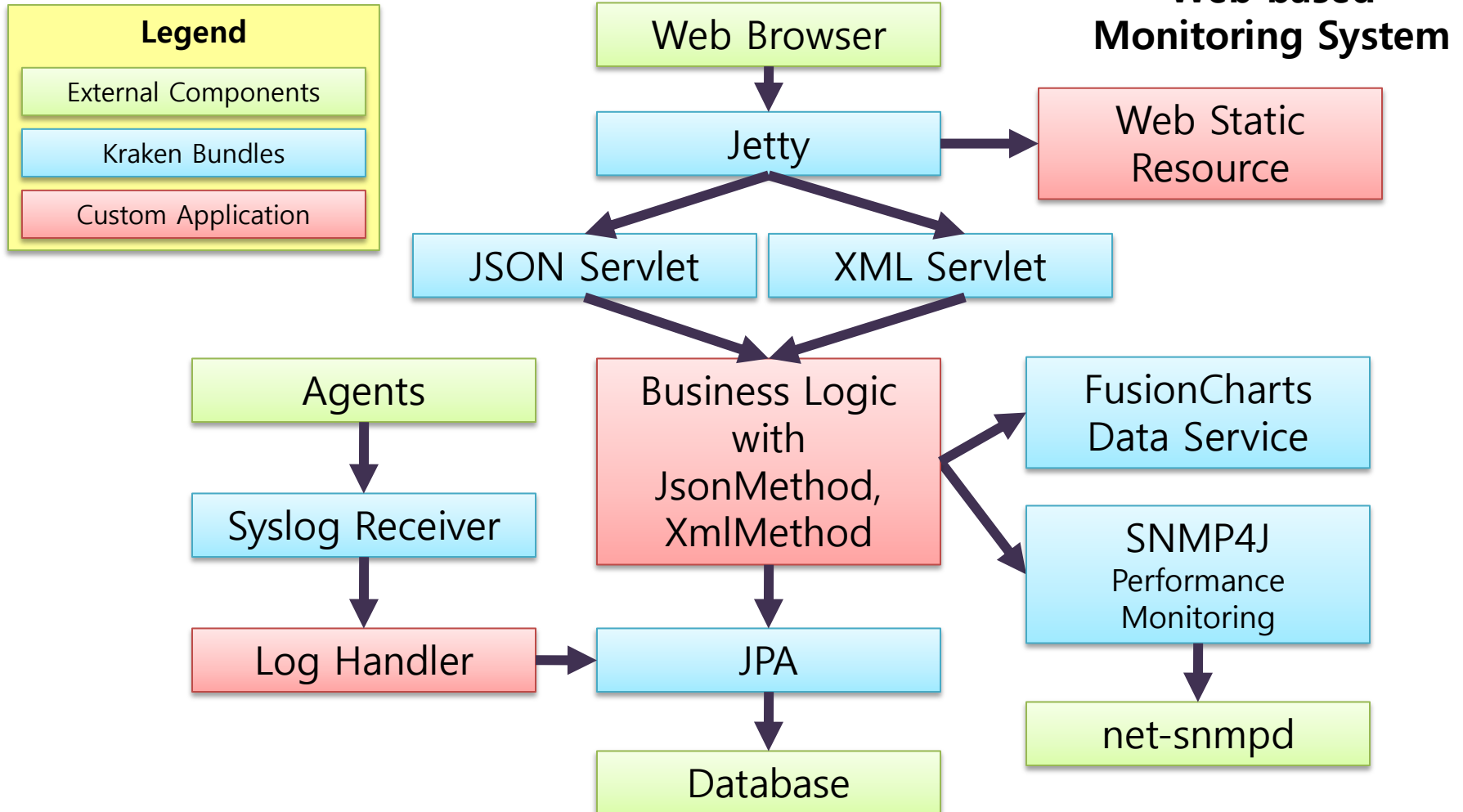
- Lack of configurability
 - Security appliances should provide runtime configuration through CLI
 - Dynamic update and version control are also needed
- Lack of integration
 - There are well-known open source libraries but you should write glue codes
 - There are well-known use cases but no simplified API provided
- Lack of reusable and standardized security components
 - Vendors use their own log formats
 - Everyone writes log parsers again and again
 - Response engine requires log merging or compression
 - Maybe you don't want to send hundreds of alarm mails per second
 - Poor interoperability
 - Data model and API can be standardized per product group

○ Kraken

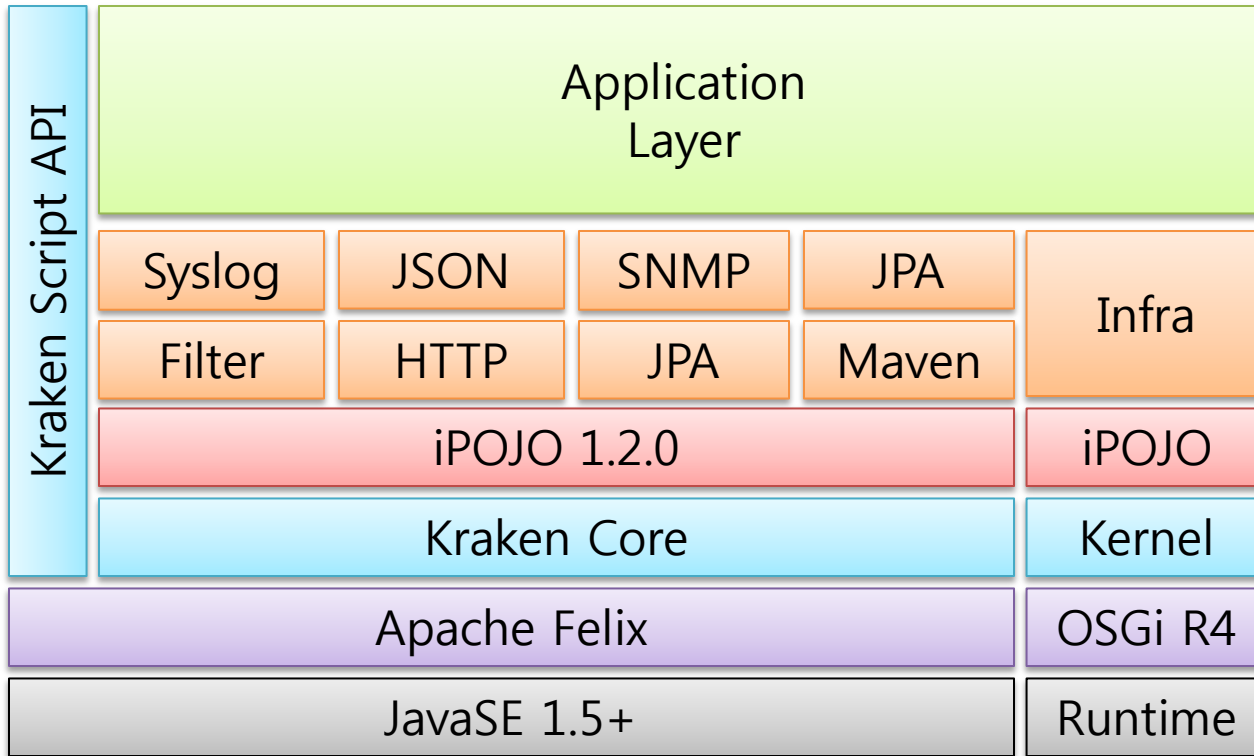
- OSGi based security platform
 - Highly configurable
 - Interactive Console
 - Hot Deploy
 - Version and Dependency Control
 - Component based Development
 - Using iPOJO (injected POJO)
 - Managed Lifecycle
 - Declarative Services (e.g. Transaction, Web Service)
 - Prebuilt Infrastructure and security components
 - Infrastructure: Filter, HTTP, JPA
 - Web services: JSON, XML, Text, FusionCharts Servlets
 - Networking: Syslog, SNMP, JPCAP, DNS
 - and more
- Distributed under Apache Software License 2

Introduction

Real world example

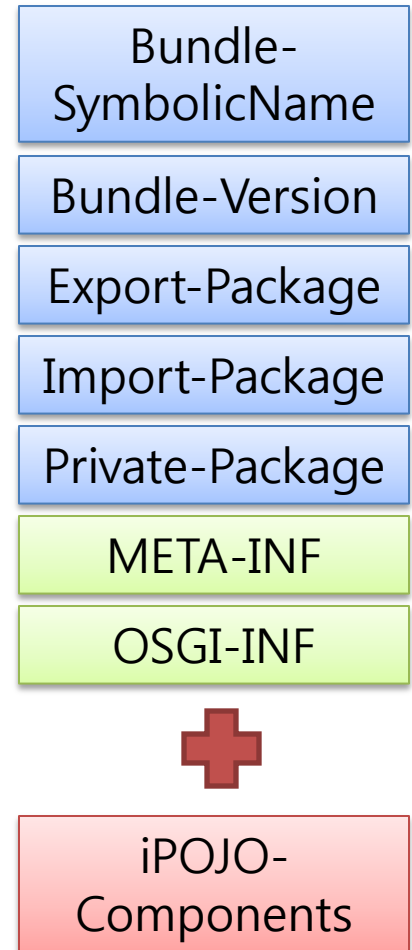


Architecture



Kraken Application Stack

OSGi Bundle



○ Reference

– AirSCAN (Wavesoft Inc.)

- Monitor and block wired, wireless, adhoc, wibro, hsdpa, bluetooth devices
- Register wireless devices to AirTight SpectraGuard Enterprise

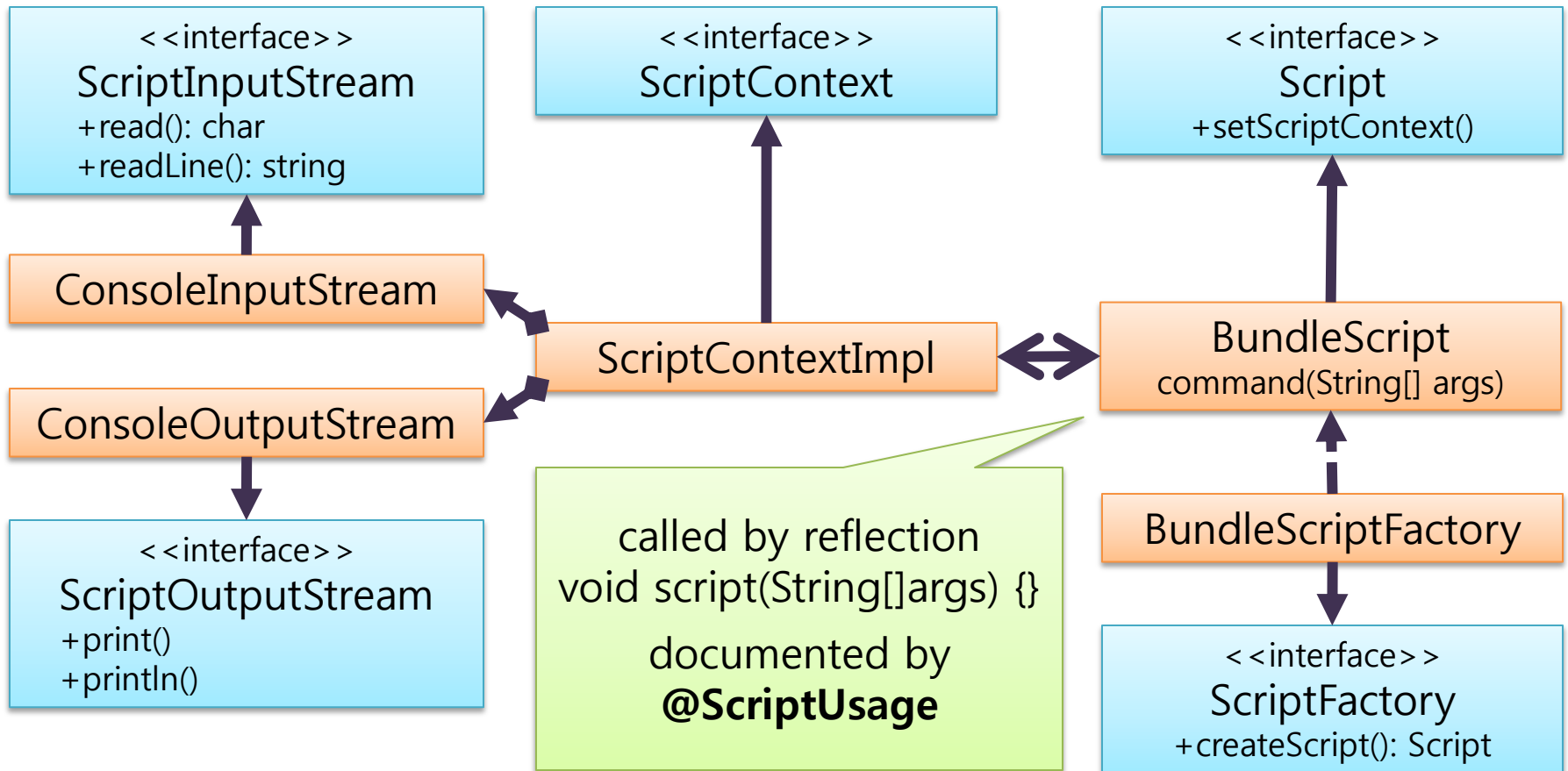
– WatchCat (NCHOVY Inc.)

- ESM as a Service
- Minimize service downtime
- Highly extensible architecture
- Under development

Kraken API

learn by example

Class diagram



○ First script example

– Create project using maven

- mvn architecture:create

- DarchetypeGroupId=org.apache.felix

- DarchetypeArtifactId=maven-ipojo-plugin

- DarchetypeVersion=1.2.0

- DgroupId=YOUR_GROUP_ID

- DartifactId=YOUR_ARTIFACT_ID

- Dversion=YOUR_VERSION

- DpackageName=YOUR_DEFAULT_PACKAGE

- Assumptions on this example:

- YOUR_GROUP_ID is org.krakenapps

- YOUR_ARTIFACT_ID is kraken-example

- YOUR_VERSION is 1.0.0

- YOUR_DEFAULT_PACKAGE is org.krakenapps.example

○ First script example

– Maven Configuration

• Edit POM file

- `<name>Kraken Example</name>`
- `<Export-Package>org.krakenapps.example</Export-Package>`
- Remove `<Private-Package>` element
- Remove `<Import-Package>` element
- Add Kraken API dependency
 - » `<dependency>`
 - `<groupId>org.krakenapps</groupId>`
 - `<artifactId>kraken-api</artifactId>`
 - `<version>1.0.0</version>`
 - `</dependency>`
- Add `<version>1.2.0</version>`
below the `maven-ipojo-plugin` element

○ First script example

– iPOJO Configuration

- Edit metadata.xml

```
<ipojo>
  <component
    className="org.krakenapps.example.ExampleScriptFactory"
    name="exampleScriptFactory"
    immediate="true"
    factory="false">
    <provides>
      <property name="alias" type="string" value="example" />
    </provides>
  </component>
  <instance component="exampleScriptFactory" />
</ipojo>
```

- All console command is prefixed with alias
 - **example.hello** means **void hello(String[] args)** method of a script created from ExampleScriptFactory
- Above configuration declares and instantiates an iPOJO component. See the [Apache Felix iPOJO Wiki documentations](#).

○ First script example

– Edit ExampleScriptFactory.java

```
package org.krakenapps.example;
```

```
import org.krakenapps.api.Script;
```

```
import org.krakenapps.api.ScriptFactory;
```

```
public class ExampleScriptFactory implements ScriptFactory {
```

```
    @Override
```

```
    public Script createScript() {
```

```
        return new ExampleScript();
```

```
    }
```

```
}
```

– ScriptFactory constructor can receive BundleContext parameter

- Finds other OSGi services and create **Script** object with them
- e.g. bundleContext.getServiceReference(interfaceName);

○ First script example

- Edit ExampleScript.java

```
package org.krakenapps.example;

import org.krakenapps.api.Script;
import org.krakenapps.api.ScriptContext;

public class ExampleScript implements Script {
    private ScriptContext context;

    @Override
    public void setScriptContext(ScriptContext context) {
        this.context = context;
    }

    public void hello(String[] args) {
        context.println("hello kraken");
    }
}
```

- Control your components at console
 - Add command methods as many as you want

○ First script example

– Build

- mvn package
- will generate `kraken-example-1.0.0.jar` in `target` directory

– Connect Kraken Console and Install

- telnet localhost 7004
- kraken> bundle.install org.apache.felix org.apache.felix.ipojo 1.2.0
download from maven central repository and install
- kraken> bundle.install file:///c:/DIRECTORY/target/kraken-example-1.0.0.jar
- kraken> bundle.list
see id number of bundles and start them
- kraken> bundle.start 1 2

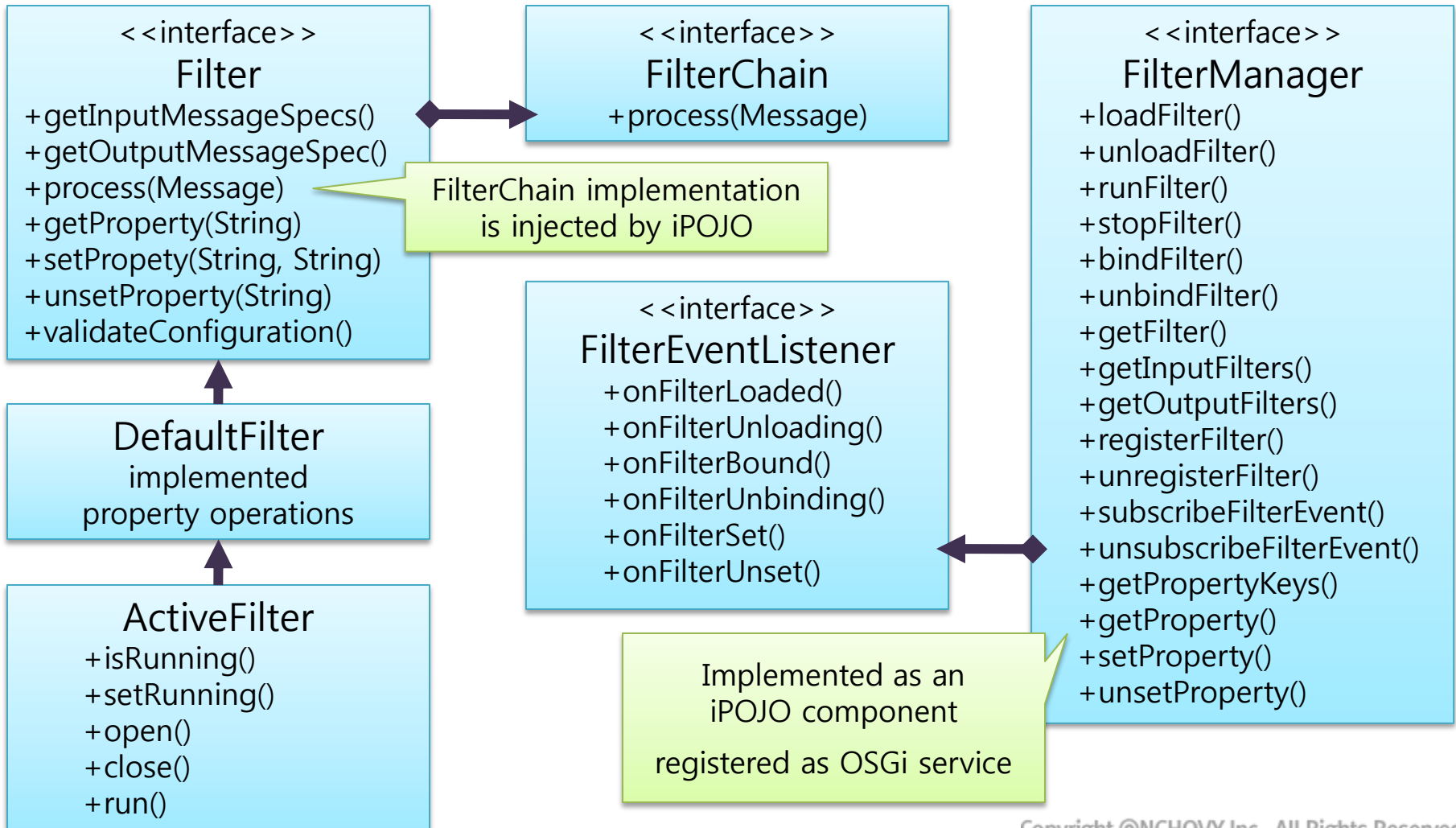
– Test

- kraken> example.hello
hello kraken

Kraken Filter

runtime filter composition

Filter

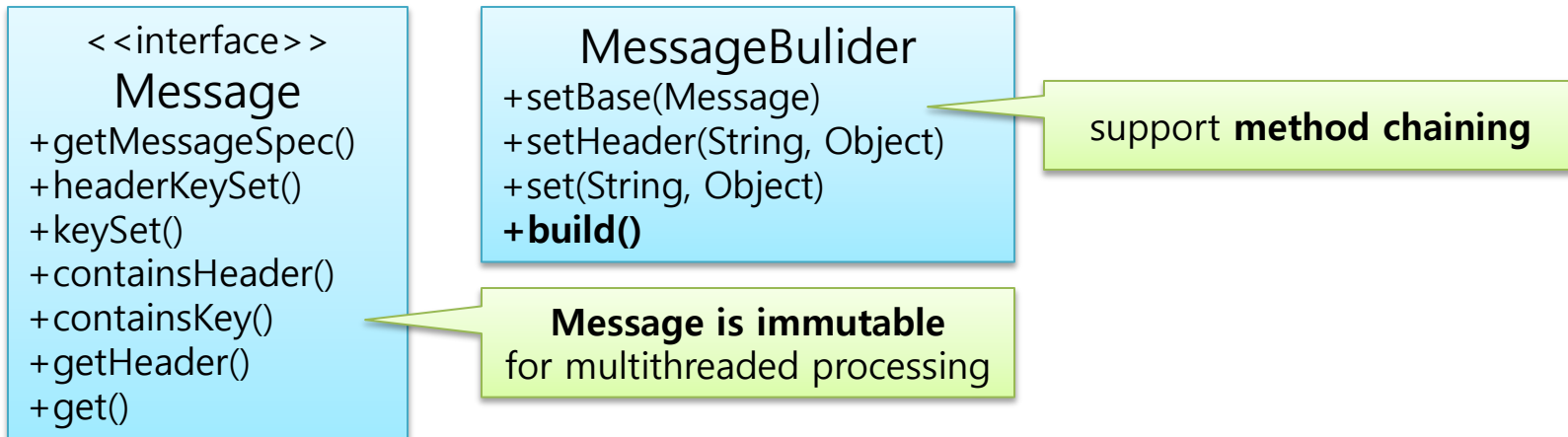


Message Specification

- Source's output spec and destination's input spec have to be matched.
 - Bind failed if specification does not match

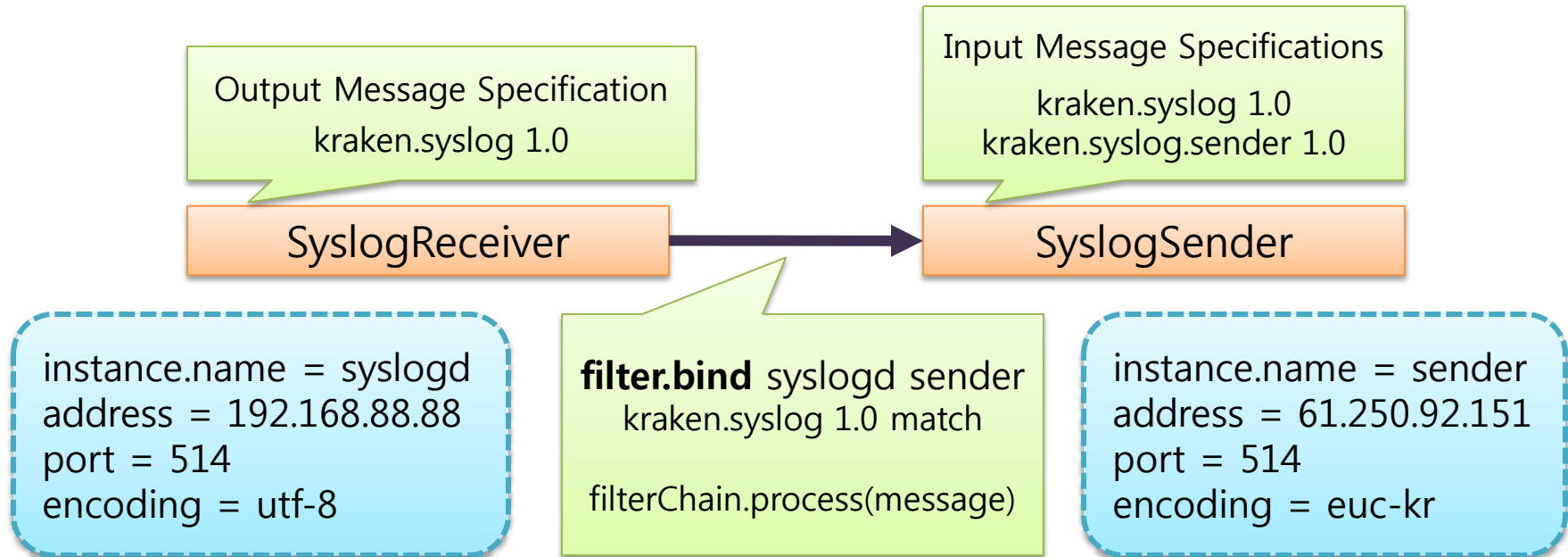


Message



Message Flow Example

- Provides complex functionality by runtime filter binding



Syslog Relay with transcoding

○ Filter Script

– backed by filter manager

- filter.list
- filter.load [filter class name] [alias]
- filter.unload [alias]
- filter.bind [source alias] [destination alias]
- filter.unbind [source alias] [destination alias]
- filter.status [alias]
 - show properties and bind status of the filter
 - list all loaded filter instances if alias is omitted
- filter.run [alias] [interval]
 - only for active filter
 - 1second if interval is omitted
- filter.stop [alias]
 - only for active filter
- filter.set [alias] [key] [value]
- filter.unset [alias] [key]

- Syslog example
 - to be continued