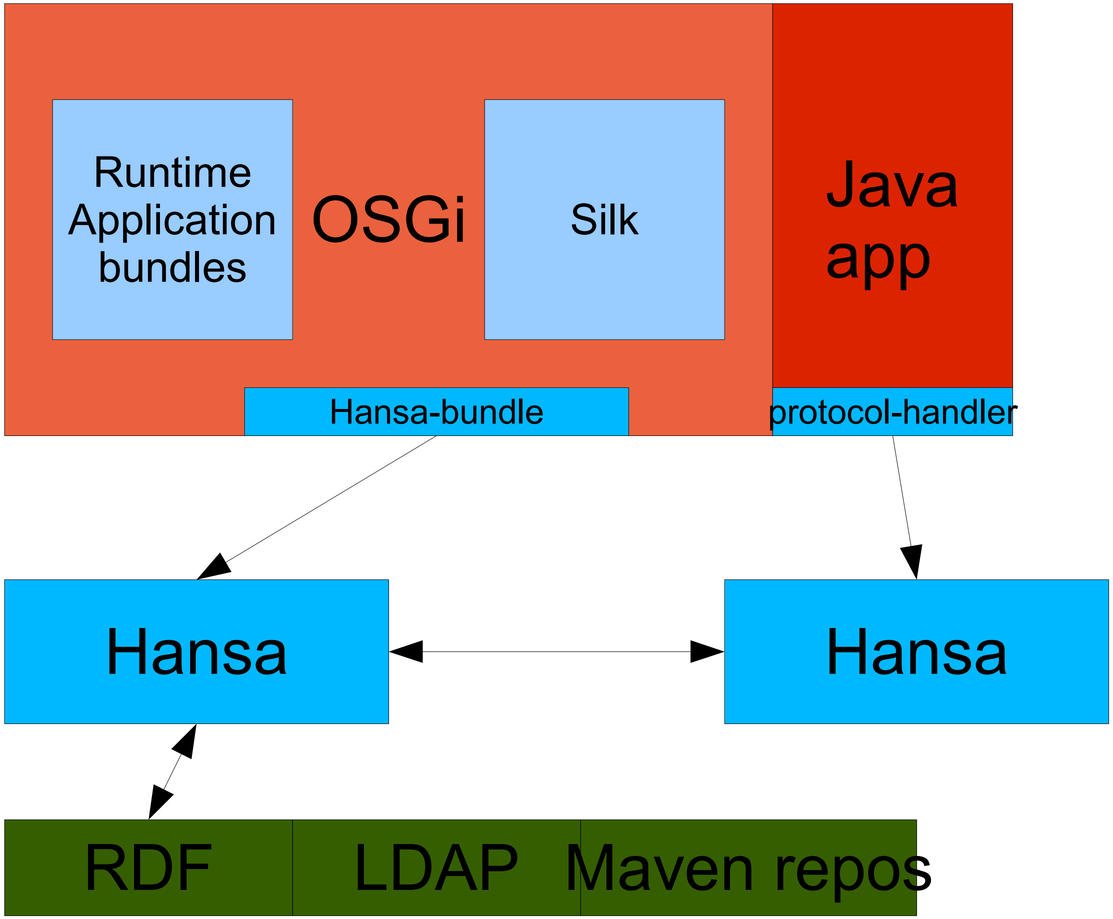
Component Oriented Programming at OPS4J

Peter Neubauer
Andreas Ronge
Niclas Hedhman

OPS4J

- Open Participation Software for Java
- Community
  - “Wiki brought to coding”
- Technically
  - Realise the full potential of COP through
    - development tools around the OSGi platform
    - OSGi components
- Commercial
  - Provide an easy way to push out commercial code into OSS based on drop-in Components

The OPS4J COP stack



Content

- The development process
  - The runtime - OSGi
  - Describe it – RDF
  - Store it – Artifact handling and component discovery
  - Build it - Silk

OSGi

- The **O**pen **S**ervices **G**ateway **I**nitiative is an non-profit corporation – name no longer applies
- OSGi produce specifications for the OSGi Service Platform:
  - A standard for components that needs to be loaded/unloaded at run-time
- OSGi was formed in 1999 at the initiative of IBM, Sun, Ericsson, Oracle, and Nortel
- For OSGi R4 there are at least three OS implementations: Equinox (Eclipse), Felix (Apache), Knopflerfish (Gatespace)

OSGi Member Companies

4DHomeNet, Inc.
Acunia
Alpine Electronics Europe Gmbh AMI-C
Atinav Inc.
BellSouth Telecommunications, Inc.
BMW
Bombardier Transportation
Cablevision Systems
Coactive Networks
Connected Systems, Inc.
Deutsche Telekom
Easenergy, Inc.
Echelon Corporation
Electricite de France (EDF)
Elisa Communications Corporation
Ericsson
Espial Group, Inc.
ETRI France Telecom
Gatespace AB
Hewlett-Packard
IBM Corporation
ITP AS

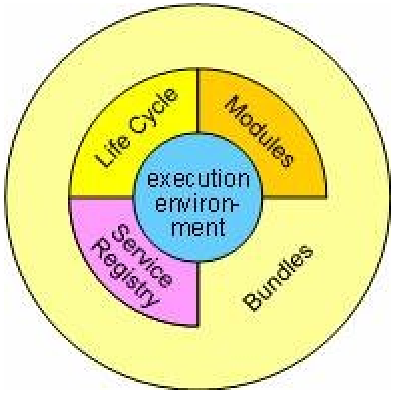
Jentro AG KDD R&D Laboratories Inc.
Legend Computer System Ltd.
Lucent Technologies
Metavector Technologies
Mitsubishi Electric Corporation
Motorola, Inc.
NTT
Object XP AG
On Technology UK, Ltd
Oracle Corporation
P&S Datacom Corporation
Panasonic Patriot Scientific Corp. (PTSC)
Philips ProSyst Software AG
Robert Bosch Gmbh
Samsung Electronics Co., LTD
Schneider Electric SA
Siemens VDO Automotive
Sharp Corporation Sonera Corporation

Application areas

- Service gateways
- Cars,
- Mobile telephony,
- Industrial automation,
- Building automation,
- PDAs,
- grid computing,
- white goods (e.g. by Bosch und Siemens)
- entertainment (e.g. iPronto),
- fleet management,
- IDEs.

The Framework

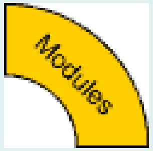
- The Framework is divided in a number of layers.
  - L0: Execution Environment
  - L1: Modules
  - L2: Life Cycle Management
  - L3: Service Registry



Execution Environment

- The L0 Execution environment is the specification of the Java environment.
  - E.g. Java 2 Configurations and Profiles, like J2SE, CDC, CLDC, MIDP etc.

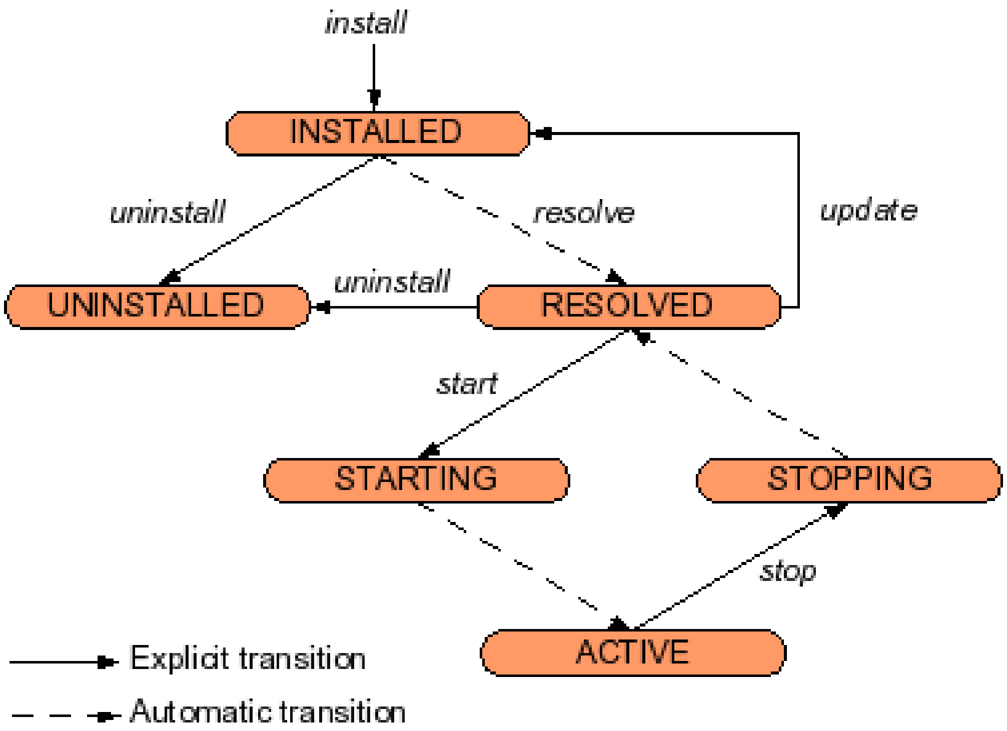
L1: Modules (bundles)



- Defines the class loading policies.
- The OSGi Modules layer adds private classes for a module as well as controlled linking between modules.
- Import/Export of packages
  - Versioning
  - Resolution of multiple package instances
  - Pooling of libraries
  - Separation of API/Implementation

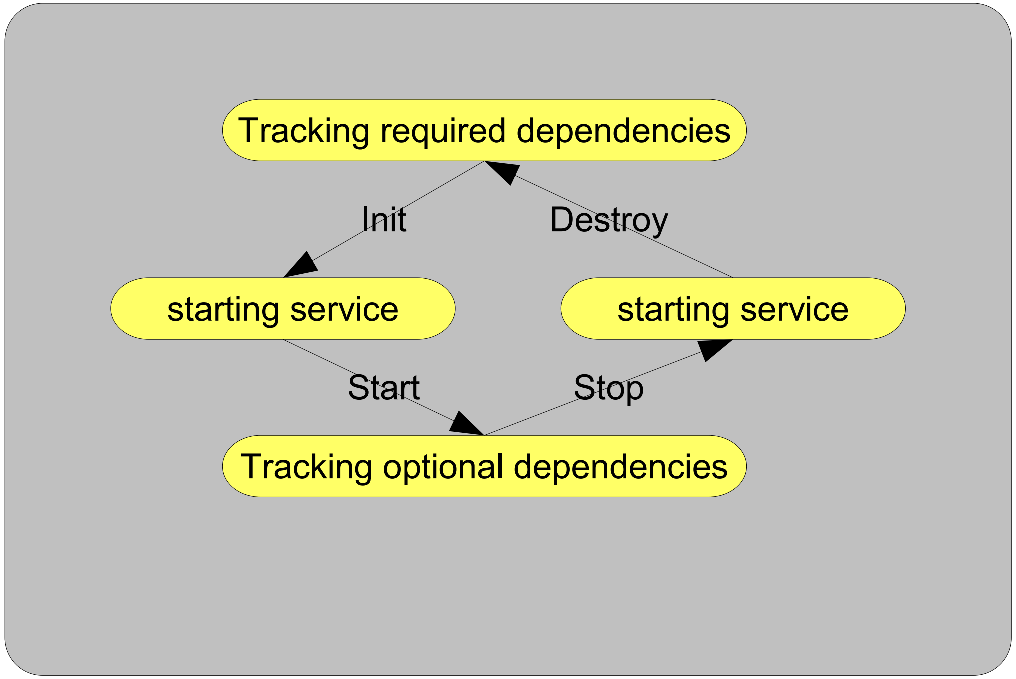
L2: Life Cycle Management

- Enable bundles to be installed, started, stopped, updated and uninstalled.



L2: Life Cycle Management

- Before activating a service, track its dependencies (imported packages)
- Handling of non-available services expected
- R4 introduces “required bundles” that are guaranteed to stay

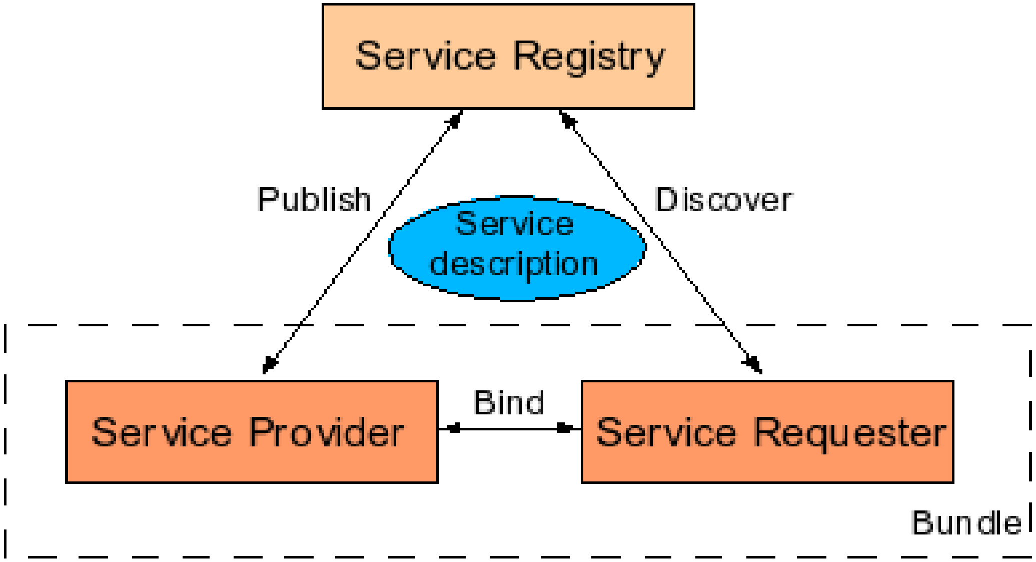

Tracking required dependencies
Init
Destroy
starting service
starting service
Start
Stop
Tracking optional dependencies

L3: The OSGi Service Platform

- Consists of
  - OSGi framework, defines for example:
  - application life cycle model
    - service registry
    - Standard Service definitions, e.g.
    - Log, Configuration management, Preferences, Http Service (runs servlets), XML parsing, Device Access, Package Admin, Permission Admin, Start Level, User Admin, IO Connector, Wire Admin, Universal plug-and-play (UPnP).
    - Implementation of these services are optional

L3: Service Registry

- Provides a cooperation model for bundles that share services that comes and goes
  - Events are defined to handle the coming and going of services. Services are Java objects.

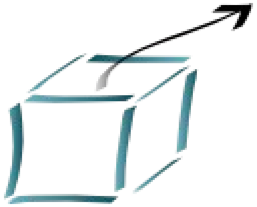


L3: What is a Service ?

- A Java class or interface
  - A.k.a. the Service interface
- with Service Properties
  - Name and Value pair
  - Allow different service providers that provide services with the same service interface to be differentiated.
- that are part of a **bundle.**
- optional: ServiceFactory
  - allow custom discovery policies

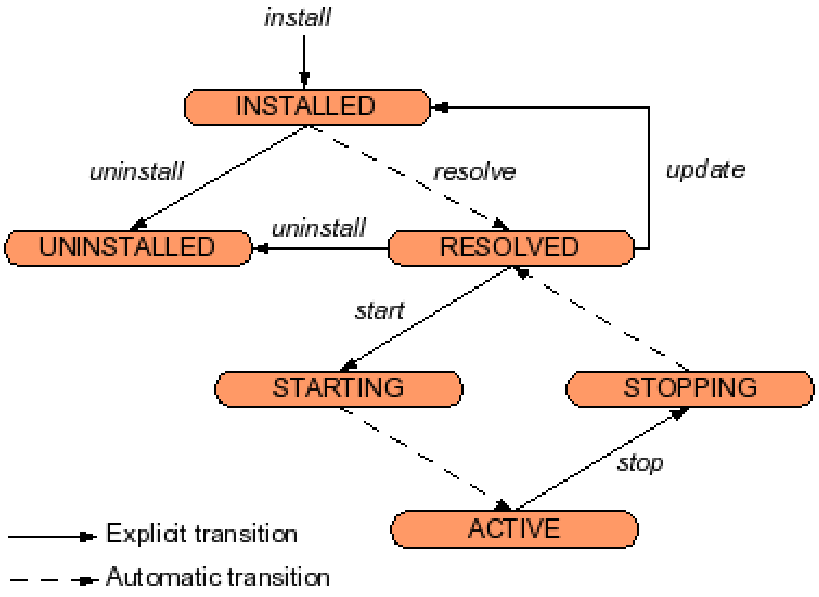
Services and Bundles

- What is a bundle ?
  - Jar file containing
  - Manifest (manifest.mf)
    - classes, other resources (pics etc.)
    - other .jar files
  - Activator implementation
  - library bundles, purely providing exported classes
    - clients can't track their reload
- What is it responsible for ?
  - Providing implementation of Service interfaces
  - Run-time service dependency management activities
    - publication, discovery and binding
    - adapting to changes resulting from dynamic availability (arrival or departure) of services that are bound to the bundle.



Deployment of bundles

- Each bundle correspond to one JAR file, contains:
  - code and resources (i.e., images, libraries)
  - The Manifest file - contains information about the bundle
- Deployment via URL pointing to bundle .jar
- Deployment activities are realized according to a well defined series of states



Deployment

- The manifest is packaged into a JAR file along with the Java class file
- The whole JAR package is actually referred to as a *bundle*.
- The manifest.mf

```
Bundle-Activator: tutorial.example1.Activator
Bundle-Name: Service listener example
Bundle-Description: A bundle that displays messages at
   startup and when service events occur
Bundle-Vendor: Richard Hall
Bundle-Version: 1.0.0
```

A Service Example

- An example of a service interface:

```
package tutorial.example2.service;
```

Bundle will share this package

```
/**
 * A simple service interface that defines a dictionary service.
 * A dictionary service simply verifies the existence of a word.
**/
public interface DictionaryService
{
  /**
    * Check for the existence of a word.
    * @param word the word to be checked.
    * @return true if the word is in the dictionary,
    * false otherwise. **/
  public boolean checkWord(String word);
}
```

The BundleActivator and Service Implementation

```
package tutorial.example2;
import …
public class Activator implements BundleActivator {

  public void start(BundleContext context) {
    Properties props = new Properties();
    props.put("Language", "English");
    context.registerService( DictionaryService.class.getName(),
      new DictionaryImpl(), props);
   }
   public void stop(BundleContext context) {
        // NOTE: The service is automatically unregistered. }

  private static class DictionaryImpl implements DictionaryService
   {
        public boolean checkWord(String word) {
               // the implemenation …
        }
   }
}
```

This package will not be shared

Registers the Service

Implementation of the service

Service provider manifest.mf

```
Bundle-Activator: tutorial.example2.Activator
Export-Package: tutorial.example2.service
Bundle-Name: English dictionary
Bundle-Description: A bundle that registers an English dictionary
   service
Bundle-Vendor: Richard Hall Bundle-Version: 1.0.0
```

One package is exported !

Service consumers manifest.mf

```
Bundle-Activator: tutorial.example3.Activator
Import-Package: tutorial.example2.service
Bundle-Name: Dictionary client Bundle-Description: A bundle that
   uses the dictionary service if it finds it at startup
Bundle-Vendor: Richard Hall
Bundle-Version: 1.0.0
```

Accessing the Service we defined

The service consumer

```
public class Activator implements BundleActivator {
  private BundleContext m_context = null;
  private ServiceTracker m_tracker = null;


  public void start(BundleContext context) throws Exception
    {
      m_context = context;
      Filter filter = context.createFilter("(&(objectClass=" +
        DictionaryService.class.getName() + ")" + "(Language=*))");
      m_tracker = new ServiceTracker (context, filter, null);
      m_tracker.open();
    }
  public String translate(String word)
  {
    DictionaryService dictionary =
      (DictionarySerivce)m_tracker.getService();
    if (dictionary != null)
    {
      return dictionary.checkWord(word)
    } else
    {
      return “”;
    }
  }
```

Tracks suitable Dictionary services

Some default action

Eclipse and OSGi

- In Eclipse 3.0 M6, the original Eclipse Runtime was replaced with a fully OSGi-based runtime.
- Eclipse plugins are now OSGi Bundles !
- Eclipse 3.1 OSGi implementation is the R4 specification reference implementation
- Equinox is a standalone implementation of OSGi

JSR-277 Java Module System

- The specification might be included in JDK7:
  - A distribution format (Java Module with metadata)
  - A versioning scheme for dependencies
  - A repository for storing and retrieving modules
  - Runtime support for the loading modules.
  - A set of support tools
- Overlaps with OSGi !
  - ”the versioning semantics in the OSGi R3 framework is simplistic”
  - ”it is impossible to support more than one version of shared package at runtime. ”
- Supporting this JSR
  - BEA Systems, Google, Jason Van Zyl, ASF,JBoss, Sun Microsystems etc. etc.

Summary

- OSGi is now used as one Java component model for J2ME, J2SE and J2EE applications.
- Provides a solution for
  - Versioning of packages and JAR files
  - Dependencies between packages and JAR files
  - Class loading issues
  - Starting and Stopping of Services

OSGi R4 additions (selected)

- A lot of influences from Eclipse development
- Service versioning
  - possible to have different versions of same service deployed
  - Ranges supported
  - Package granularity
- Declarative Services
  - Lazy instantiation of services
  - Framework knows of services without activating the bundle
- Extension bundles
  - make e.g. URLStreamHandler fully replaceable
  - additions to boot classpath via bundles (e.g. java.sql.*)

The artifact system (Hansa)

- The need for a system that transparently connects existing applications with repository systems
- Existing solutions limited to
  - build systems (Maven2 POM)
  - runtime solutions (Eclipse bundle manifest, EJB etc.)
- The solution: custom protocols
  - already used in Eclipse bundle:// content URLs
  - in normal Java systems
  - non-intrusive
  - application does not know about the existence of Hansa

Hansa and Artifacts

- Resources exists as Artifacts in repositories
- Artifacts can be any type
- Each artifact has an unique identifier
  - `artifact:[type]:[group]/[name]#[version]`
- Hansa access artifacts by URI
- Location independence
  - local cache
  - remote repository servers (not only file structures)
  - dynamic discovery of new servers (Jini etc)

Protocol: artifact

- Resources exists as Artifacts
- OSGi
  - registration via URLStreamHandler service
- Plain java:
  - Uses an URL protocol handler
  - JVM arguments:
    - java.protocol.handler.pkgs=org.ops4j.hansa
- Code:

```
URL url = new java.net.URL(”artifact:txt:ops4j/niclas/example#42”)
InputStream is = url.getConnection().getInputStream();
```

Protocol: link

- Links any resource to an URI
- Consumer side construct
  - suitable for libraries
  - will be linked to target URL at runtime
- Much like symlinks in Linux
- Code:

```
URI google = new URI( "http://www.google.com" );
URL linkUrl = new URL( "link:my/google" );
Class[] type = new Class[] { Link.class };
Link link = (Link) linkUrl.getContent( type );
link.setTargetURI( google );

usage:

URL google = new URL( "link:my/google" );
```

Silk – the smooth build system

- Sheets
  - versioned collection of Strands
- Strands
  - provides rule sets into the main engine
- Rules engine at the core (Drools)
  - rules are independent of each other
  - triggered via type insertion into the WorkingMemory
- Versioning of the whole build process via build sheets
  - referenced as artifacts
- All strands are OSGi bundles
  - hot redeploy for every build
  - multiple versions for multiple builds possible
  - multiple builds (and dependency builds) simultaneously
  - distributed builds

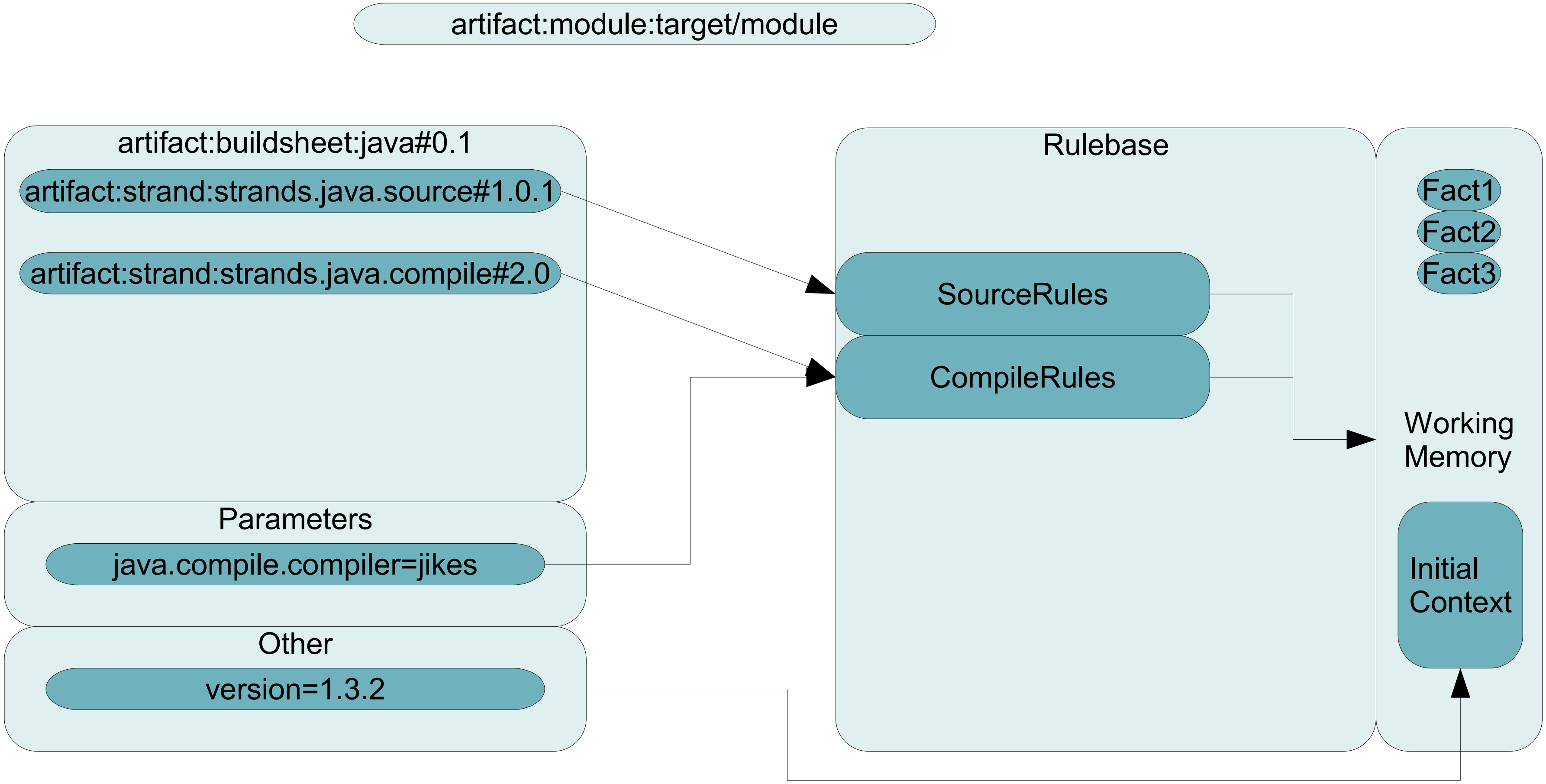
Silk – Why RDF?

- Build systems use and produce resources
- RDF = Resource Description Framework
- Descriptions for
  - Dependencies
  - Versions
  - Licenses
  - Compatibility
  - Publishers
  - Aggregations etc. etc.
- RDF is scalable (Semantic Web)
- RDF builds on URIs, transparent via Hansa
- RDF is searchable
- RDF is publishable (via Hansa)

Silk – Why Rules?

- Pure Logic can't be easily overridden (code)
- Rules can be intermixed at will, even prioritised
- Implicit rules are made visible
  - Ant: compile only if the sources are modified
- Rules engines provide possibility for DSL
  - DSL = Domain Specific Languages
  - With DSL the logic is understandable to the Domain Expert (CM)
  - special semantic language for build systems possible
  - see on that LOP (http://www.onboard.jetbrains.com/is1/articles/04/10/lop/)

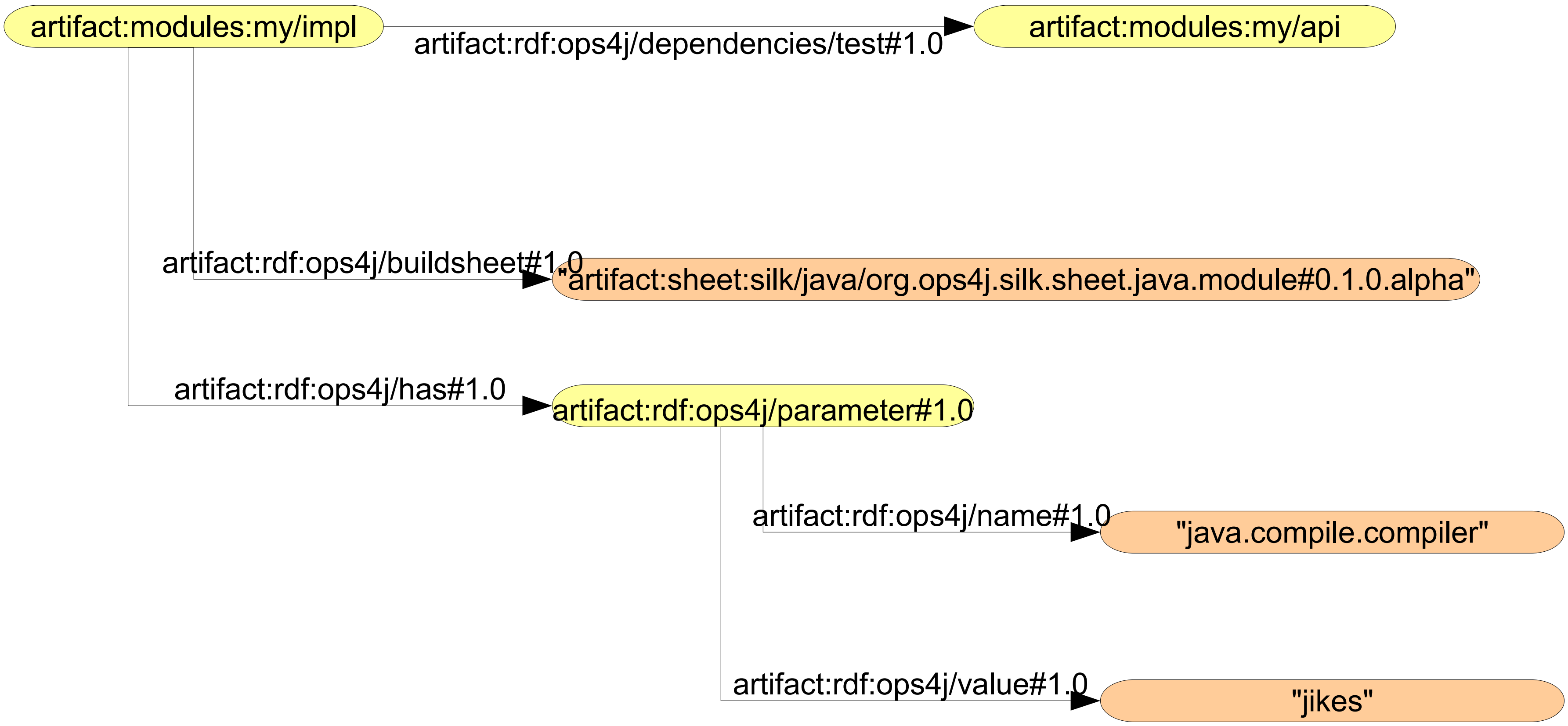
Silk – the smooth build system


artifact:module:target/module
artifact:buildsheet:java#0.1
artifact:strand:strands.java.source#1.0.1
artifact:strand:strands.java.compile#2.0
Parameters
java.compile.compiler=jikes
Other
version=1.3.2
Rulebase
SourceRules
CompileRules
Fact1
Fact2
Fact3
Working Memory
Initial Context

RDF

OSGi

RDF for component URI


artifact:modules:my/impl
artifact:rdf:ops4j/dependencies/test#1.0
artifact:modules:my/api
artifact:rdf:ops4j/buildsheet#1.0
"artifact:sheet:silk/java/org.ops4j.silk.sheet.java.module#0.1.0.alpha"
artifact:rdf:ops4j/has#1.0
artifact:rdf:ops4j/parameter#1.0
artifact:rdf:ops4j/name#1.0
"java.compile.compiler"
artifact:rdf:ops4j/value#1.0
"jikes"

Example RDF (N3)

```
@prefix ops4j:   <http://www.ops4j.org/ns/2005/silk> .
@prefix ops4j_modules:  <http://scm.ops4j.org/repos/ops4j/projects/rdftest/modules> .
<http://scm.ops4j.org/repos/ops4j/projects/rdftest/modules/module1>
      ops4j:name
              "Module1" ;
      ops4j:version
              "1.0.4.alpha" .
      ops4j:description
              "the first module" ;
      ops4j:buildsheet
              "artifact:sheet:silk/java/org.ops4j.silk.sheet.java.module#0.1.0.alpha" ;
      ops4j:dependsOn
              <http://scm.ops4j.org/repos/ops4j/projects/rdftest/modules/module2> ;
      ops4j:has
              [ a       ops4j:parameter ;
                ops4j:key
                        "java.source.location" ;
                ops4j:value
                        "src/java"
              ] ;
<http://scm.ops4j.org/repos/ops4j/projects/rdftest/modules/module2>
      ops4j:description
              "second module" ;
      ops4j:version
              "3.8.1" .
```

Silk – build sheet

```
<silk:sheet
    xmlns:silk="http://www.ops4j.org/ns/2005/silk/sheet"
>

  <silk:name>JavaModule</silk:name>

  <silk:description xml:lang="en" >
This BuildSheet builds a Java module. The following features are implemented;
 *  Picks up Java sources from any URL(s).
 *  Compiles these sources in a single pass Java compile.
 *  Picks up Java unittest (JUnit) from any URL(s).
 *  Creates a Junit report and publishes it as an artifact.
 *  Executes JavaDoc on the sources and publishes that as an artifact.
 *  Package the resulting classes and resources into a Jar and publish that as
    an artifact.
  </silk:description>

  <silk:strands>
    <silk:uri>artifact:jar:silk/java/org.ops4j.silk.strands.java.module#1.0.0</silk:uri>
    <silk:uri>artifact:jar:silk/java/org.ops4j.silk.strands.java.source#1.0.0</silk:uri>
    <silk:uri>artifact:jar:silk/java/org.ops4j.silk.strands.java.compile#1.0.3</silk:uri>
    <silk:uri>artifact:jar:silk/java/org.ops4j.silk.strands.java.junit#1.1.0</silk:uri>
    <silk:uri>artifact:jar:silk/java/org.ops4j.silk.strands.java.javadoc#1.0.0</silk:uri>
    <silk:uri>artifact:jar:silk/java/org.ops4j.silk.strands.java.jar#1.0.3</silk:uri>
    <silk:uri>artifact:jar:silk/java/org.ops4j.silk.strands.java.publish-artifact#1.0.3</silk:uri>
  </silk:strands>

</silk:sheet>
```

Silk – example ruleset (compile)

```
<?xml version="1.0" encoding="UTF-8" ?>

<rule-set name="java compile rules"
         xmlns="http://drools.org/rules"
         xmlns:java="http://drools.org/semantics/java"
         xmlns:groovy="http://drools.org/semantics/groovy"
         xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
         xs:schemaLocation="http://drools.org/rules rules.xsd
      http://drools.org/semantics/java java.xsd">


         <rule name="Compile Raw Java Sources">
```

Parameters

```
                   <parameter identifier="params">
                            <class>org.ops4j.silk.strands.java.compile.Parameters</class>
                   </parameter>
                   <parameter identifier="source">
                            <class>org.ops4j.silk.strands.java.source.RawSourceFact</class>
                   </parameter>
               <java:condition>source.getLastModified()>new
File(“target”).lastModified()</java:condition>
                   <groovy:consequence>
                   System.out.println("compiling: " + source.getMainLocation());
                            def sourceDir = new File(source.getMainLocation());
```

Action

```
                            def ant = new AntBuilder();
                            def targetDir = 'classes';
                            ant.mkdir(dir:targetDir);
                            ant.javac( srcdir:sourceDir.getAbsolutePath(), destdir:targetDir,
compiler:params.getCompilerType() );
                            drools.assertObject(new RawCompileFinishedFact());
                   </groovy:consequence>
         </rule>
</rule-set>
```

Resources

- OPS4j
  - http://www.ops4j.org
  - general@lists.ops4j.org
- OSGi
  - http://www.osgi.org
  - http://eclipsercp.org/
  - http://eclipse.org/equinox/documents/osgicongress2005/mcaffer_1012_1530.pdf
- Drools
  - http://drools.codehaus.org
- JSR 277
  - http://www.jcp.org/en/jsr/detail?id=277