Enhancing OSGi with Explicit, Vendor Independent Extra-functional Properties

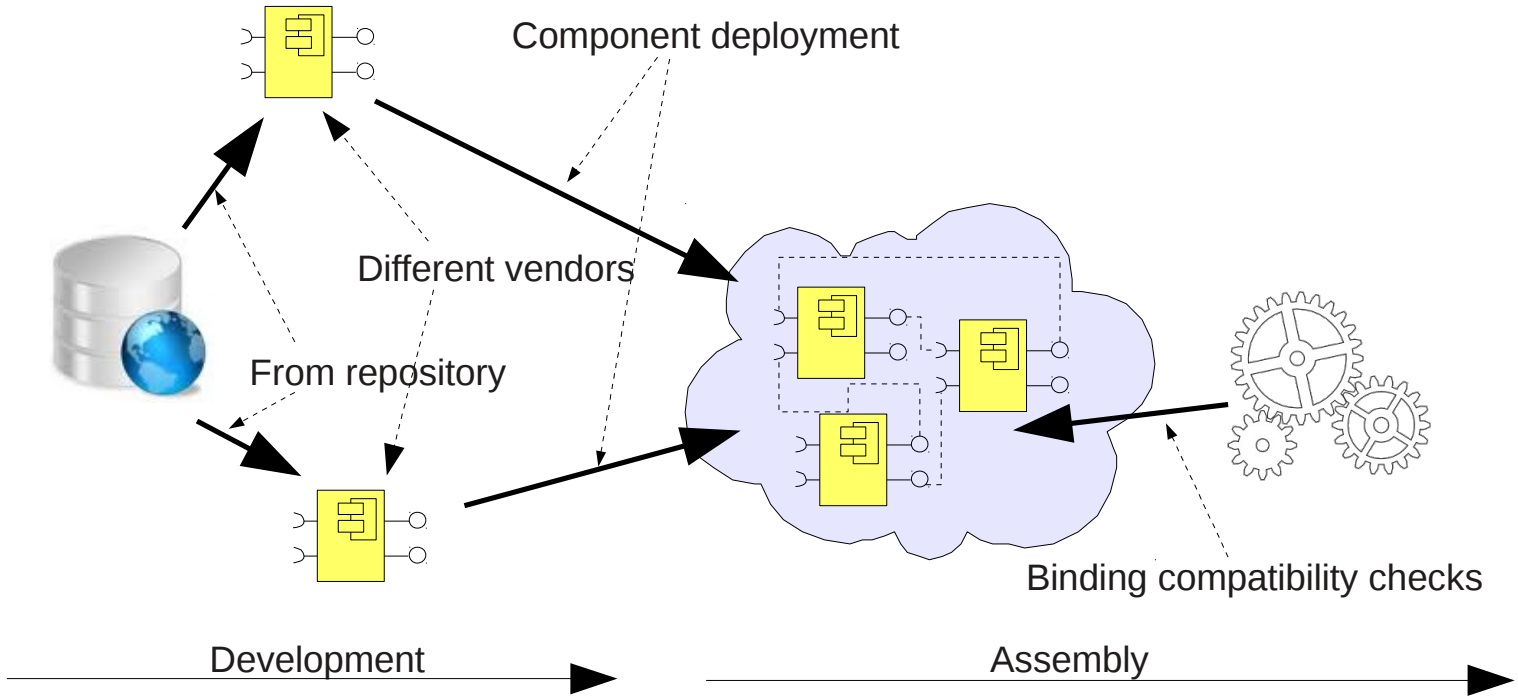
Kamil Ježek Premek Brada Lukáš Holý
Department of Computer Science and Engineering
University of West Bohemia
Univerzitni 8, 30614 Pilsen, Czech Republic
{kjezek|brada|lholy}@kiv.zcu.cz

TOOLS 2012

Agenda

- Component-based programming + Extra-functional properties
- Semantics weak extra-functional properties in OSGi
- Proposed semantic rich extra-functional properties
- Application to OSGi

Motivation



- Component independently developed and deployed
- Different groups of people involved
- Compatibility verification more important than before
- Extra-functional properties should be considered

What are Extra-functional Properties?

No common understanding:

- Palladio simulates systems to asses performance and reliability
- ProCom defines parametrised values
- PECOS composes Petri-nets to model timing and synchronisation

In this work:

> *An extra-functional property holds measurable values, explicitly provided with a software system, to specify a characteristic of the system apart from its genuine functionality, to enrich client's understanding of the usage of the system, supported by technical [computational] means.*

E.g.: `System A: Require - network_speed = 10Mb/s`

OSGi Overview

OSGi is:

- Industrial standard to modularised systems
  - Modules (components) Java jar files – called Bundles
  - Explicit lifecycle of Bundles from Install and Start to Stop and Uninstall
- Strong isolation of each Bundle (on classpath level)
  - Communication (among others) via explicitly registered services
  - Only exported packages are accessible
  - Better isolation control than Java coarse grained access modifiers
  - Provided and required features explicitly stated in manifest file

OSGi Parametrised Packages

Parametrised packages

```
Export-Package: cz.zcu.kiv.web.html;
  version=1.3,html_version=5.0

Import-Package: cz.zcu.kiv.web.html;version=1.3;
  resolution:=optional,html_version=5.0
```

- `version` – build-in parameter
- `html_version` – user defined parameter

Drawbacks:

- No explicit semantics and unification of user defined parameters (e.g. `html_version` vs. `version_html`)
- Different vendors do not have to even know existence of these parameters

OSGi Capabilities

Capabilities:

```
Provide-Capability: cz.zcu.kiv.web.ui;ui_library=Qt

Require-Capability: osgi.ee;
  filter:="(|(ui_library=GTK)(ui_library=Qt))"
```

- `osgi.ee` – build-in name-space
- `cz.zcu.kiv.web.ui` – user defined name-space

Drawback:

- Name-spaces should have proposed semantics, however, no technical mean to exchange user defined ones among developers is provided

OSGi Parametrised Services

Parametrised services:

```
HashTable ht = new HashTable();
  ht.put("network_speed", 10);
  bundleContext.registerService(Net.class, this, ht);

services = bundleContext.getServiceReferences(
  Net.class, "(network_speed >= 10)");
```

Drawback:

- Developers must study source-code or run bundles to determine services parameters

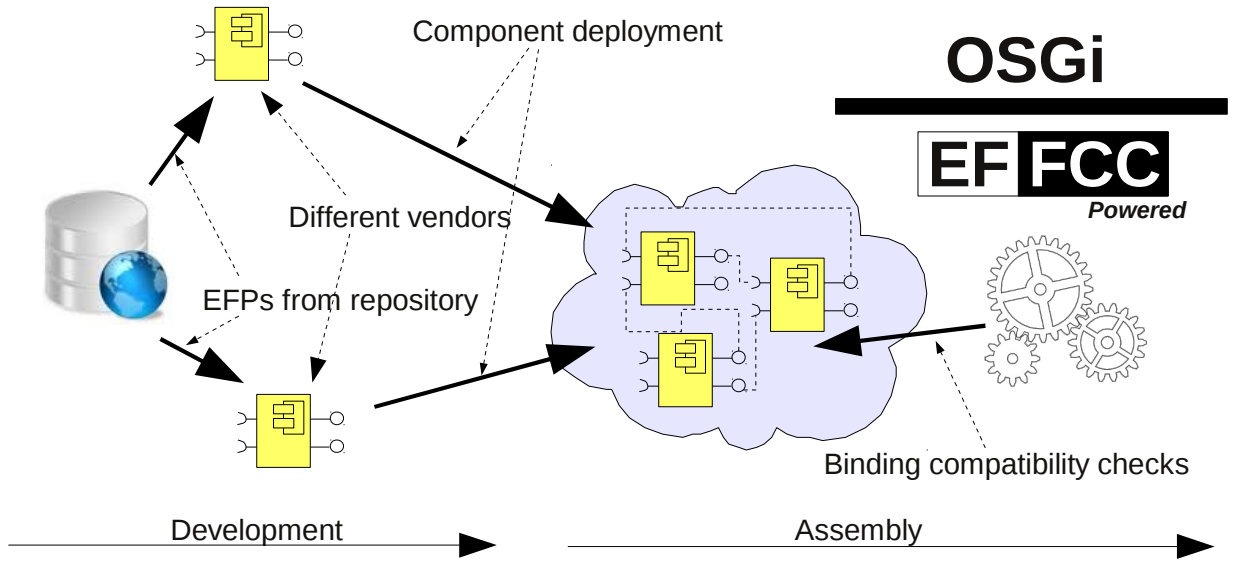
Declarative Services:

- Services defined in XML files
- Solves need to study source-code
- Semantics of parameters still weak

Proposed Solution

EFFCC – Extra-Functional properties Featured Compatibility Checks

- Explicit definitions of properties in repository
  - Same properties shared by different developers
- Properties grouped by application domain
  - Semantics assigned for each domain
  - Sub-domains supported



EFFCC Example and Formalisation

$$e = (n, E_d, \gamma, V, M)$$
$$GR = (id, name, \{e_i\})$$

```
462, Browser: {
network_speed,
 {}, Integer, decreasing, {fast, slow}, "Mb/s", ... }
```

$$LR = (id, GR, name, \{LR_i\}, S, D)$$

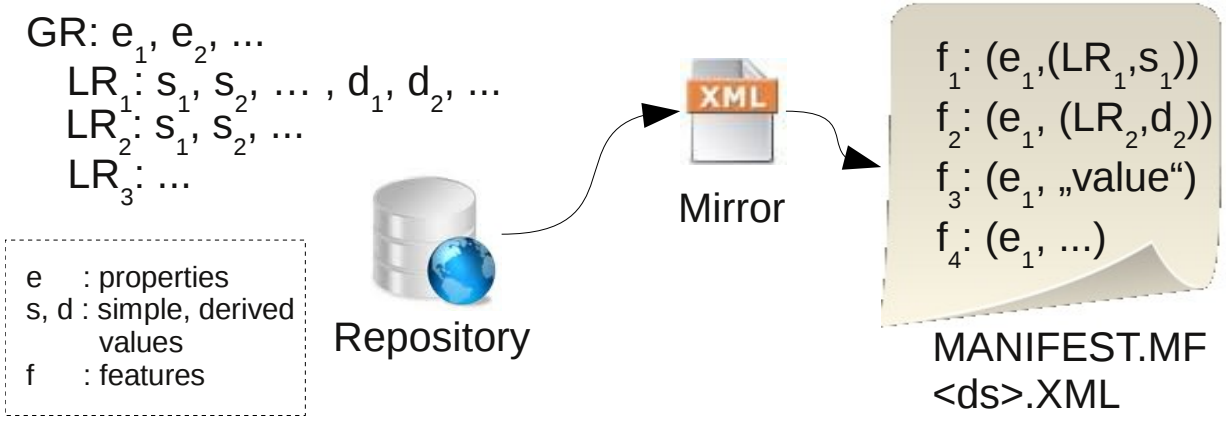
```
463, Intel Dual Core 2.8GHz {
 network_speed: slow = [1..10), fast = [10..100), ...
```

$$A = F \times E \times V_A, V_A = S \times D \times V \times computed$$

```
RenderingEngine {
 Require-Capability : 462.network_speed = 463.slow, ..
```

Enriched OSGi Parameters Semantics

- All properties stored in unified remote repository (J2EE Server)
- Actually used properties mirrored in XML file, distributed with Bundles
- XML contains semantic rich properties
- Original OSGi parameters, attributes and filters linked with XML files



Attributes, Parameters and Filters Format

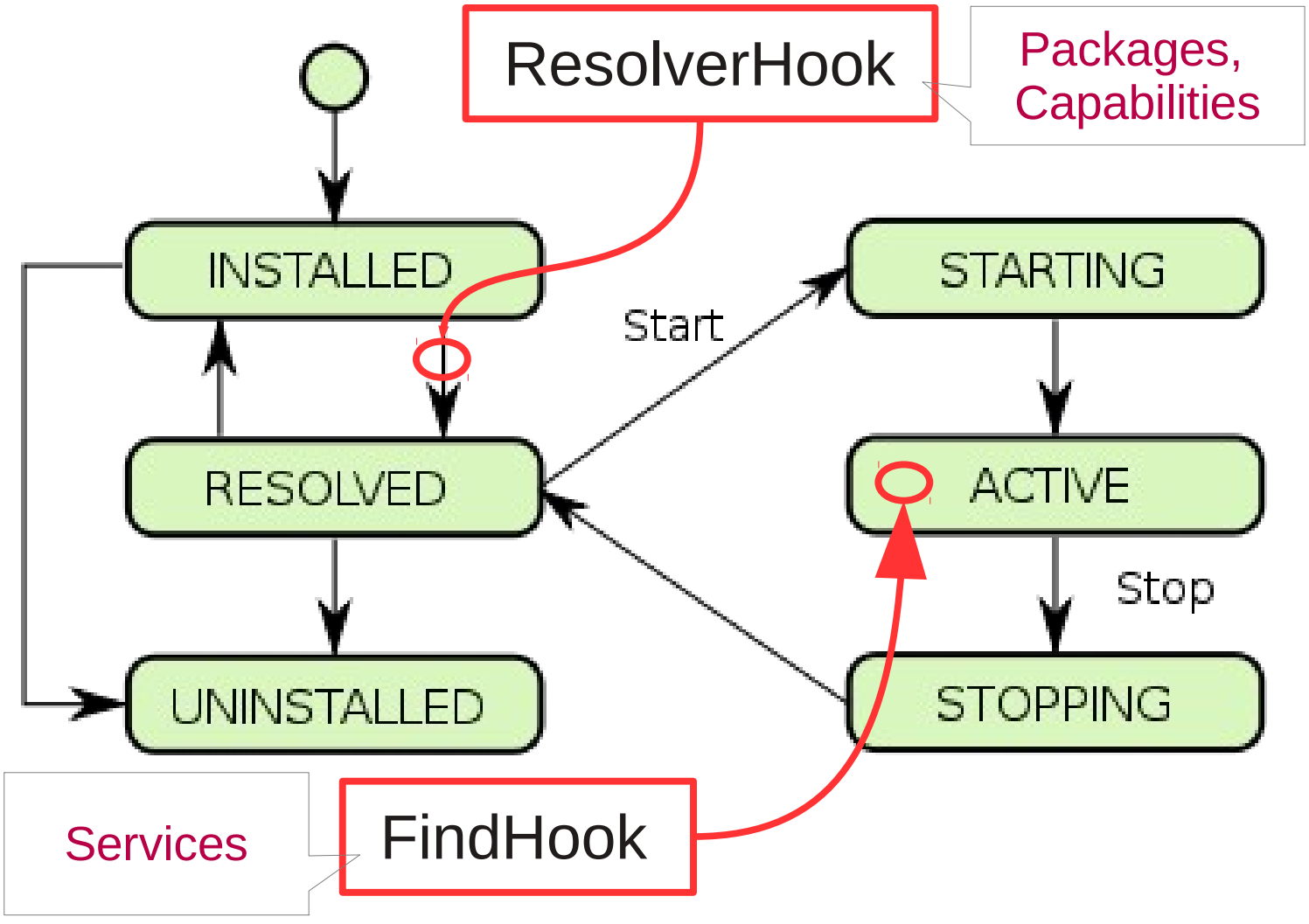
New proposed format to write down OSGi attributes and filter:

```
<gr-id>.<efp>=<lr-id>.<value>
```

- `<gr-id>` – link to XML domain definitions part
- `<lr-id>` – link to XML sub-domain part
- `<value>` – value from XML, original OSGi value or formula

E.g.:
Original OSGi: `network_speed = 10`
Enriched OSGi: `462.network_speed = 463.slow`
`462.dhtml = 465.computed`, $true \Leftrightarrow js_{ver} > 0 \& html_{ver} \geq 4$

Enriched OSGi Life-cycle



Implementation

Two hook methods:

```
ResolverHook:
void filterMatches(BundleRequirement requirement,
Collection<BundleCapability> candidates)
```

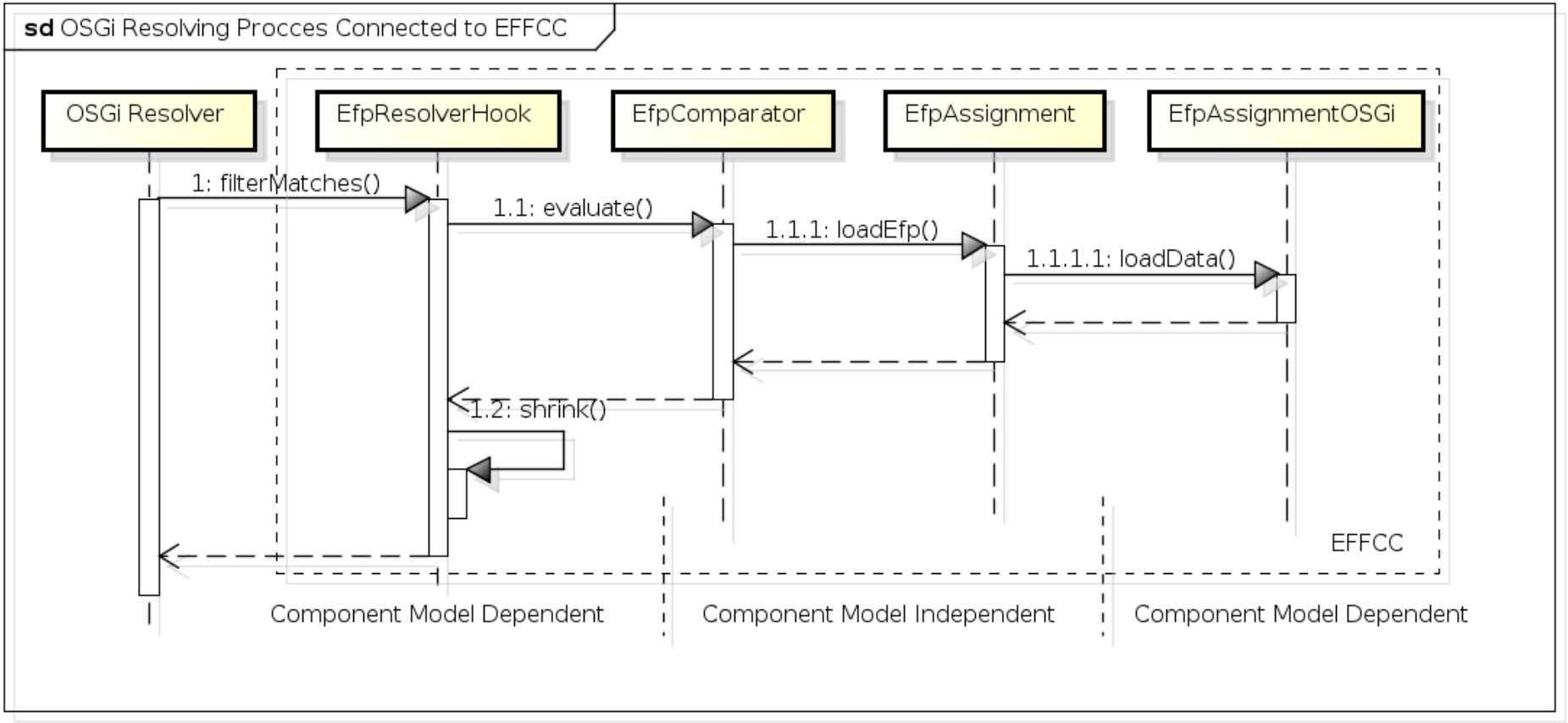
- It represents set of candidates fulfilling requirement

```
FindHook:
void find(BundleContext context, String name,
  String filter, boolean allServices,
  Collection<ServiceReference<?>> references);
```

- It represents set of services fulfilling name and service filter

In both cases, candidates can be removed according to additional compatibility checks

Implementation (II)



Conclusion

- OSGi allows ad-hoc extra-functional properties definitions
- Package export/import allows only comparison to equality (replaceable by capabilities)
- Limited number of data types (Number, String, Version)
  - User-defined types may be added. Unfortunately, not implemented in Apache Felix and Equinox (solution: external “manifest”)
  - Problem to evaluate e.g. “slow” vs. “fast” – OSGi compares as Strings
- Generic extra-functional properties mechanism proposed
  - Mechanism applied to OSGi
  - OSGi extended but not modifications
  - Declarative services suggested

Thank you for your attention

Contact us:
{kjezek|brada|lholy}@kiv.zcu.cz
Research group: http://www.kiv.zcu.cz/research/groups/dss/
EFFCC project: http://www.assembla.com/spaces/show/efps