Practical Verification of Component Substitutability using Subtype Relation

Premek Brada, Lukas Valenta
Dept of Computer Science
University of West Bohemia
Pilsen, Czech Republic

Goal and contents

- Show how subtyping can be used as substitutability check for components
- Show applicability on industrial platform
- Hint problems associated

- Constraints
  - working with mainstream platforms (EJB, OSGi)
  - use distribution form of component only

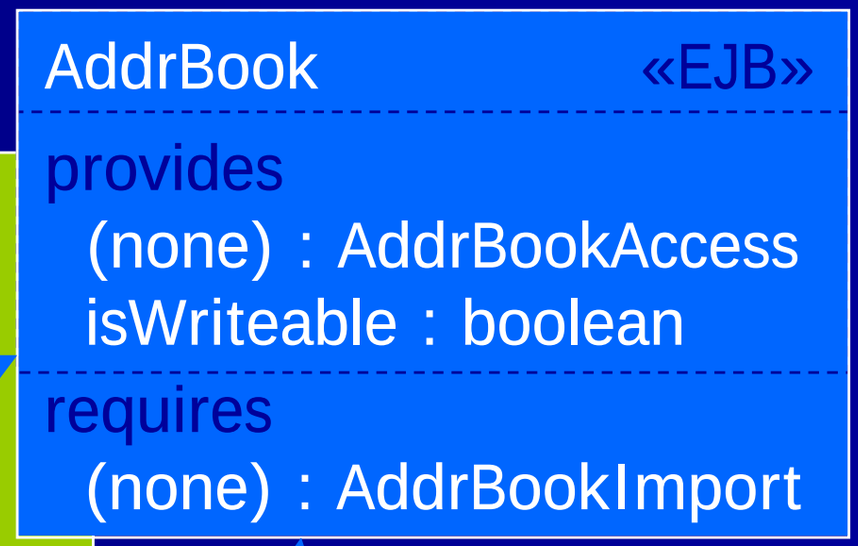
Assignment analogy

- *Vehicle v := (Car) ford;*
  - triggers type check
  - dynamic type checking allows unforeseen subtypes

- Basic (simple) idea for components:
  - take component type representation
  - perform subtype check
  - allow/ban substitution

Component specification as type

- Whole component interface representation
- Type seen as set of elements
  - provided/required “sides”




```
<enterprise-beans>
 <session>
   <ejb-name>AddrBook</ejb-name>
   <remote>hello.beans.AddrBookAccess</remote>
   <session-type>Stateless</session-type>
   ...
   <ejb-ref>
          <ejb-ref-name>ejb/AddressBookData</ejb-ref-name>
          <ejb-ref-type>Entity</ejb-ref-type>
          <remote>hello.beans.AddrBookImport</remote>
   </ejb-ref>
 </session>
</enterprise-beans>
```

```
public interface AddrBookAccess {
    public String listAddrBook(int id);
    public Address getAddressInBook(int idBook, int idAddress);
}
```

Substitutability using subtype

Subtype checks: expressing the results

- Type differences evaluated in subtype check
- Classification

  $diff(a,b) = spec \Leftrightarrow b <: a$

  – $none < ins \mid del < spec \mid gen < mut$

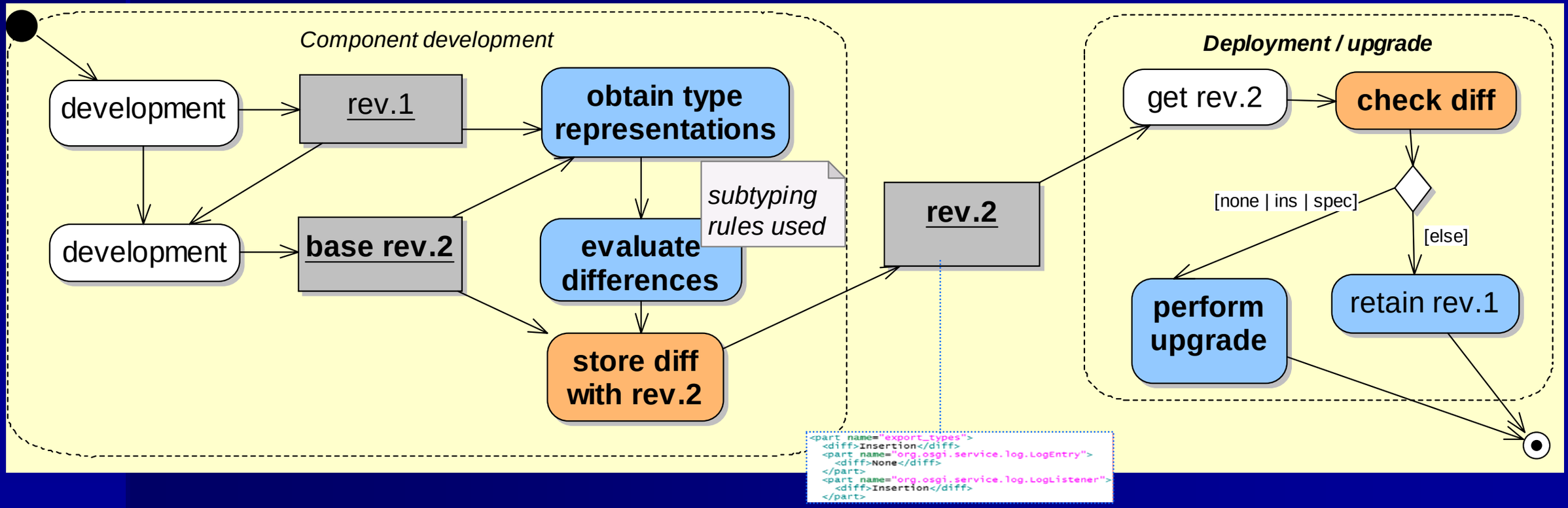
  – evaluate at leaves, combine towards root

    - $ins \oplus del \rightarrow mut$, $ins \oplus spec \rightarrow spec$, …

```
<part name="export_types">
  <diff>Insertion</diff>
  <part name="org.osgi.service.log.LogEntry">
    <diff>None</diff>
  </part>
  <part name="org.osgi.service.log.LogListener">
    <diff>Insertion</diff>
  </part>
```

Upgrades: a special case

- We could perform the check on component type always but...



Practical application

- Goal: safe OSGi bundle updates
- OSGi bundle
  - industry standard Java components
  - metadata in bundle manifest
    - provided + required packages declared
  - framework manages lifecycle, wiring

Compatibility checks in OSGi

- Bundle meta-data
  - includes package (→type) and version data
- Binding rules: version numbers match

major.minor.micro scheme

Bundle-SymbolicName: useradmin
Bundle-Version: 2.0.0
Export-Package: org.osgi.service.useradmin;**specification-version=1.1.0**
Import-Package: org.osgi.framework;**version="[1.2.0,2.0.0)"**

- Problem: weak semantics ⇒ no guarantee

Reliable bundle update

- Approach applied
  - store diff in bundle manifest
  - map to version
    - major++ ⇔ gen/del/mut difference
    - minor++ ⇔ spec/ins difference
- Consequence:

```
s>run_compare.bat examples\classes\log-old.jar examples\classes\lo
ar examples\classes\log-new_export-dist.jar
Difference: Specialization
Version changes:
 Export package 'org.osgi.service.log': 1.1 -> 1.1.1
 Export package 'org.osgi.service.test' is new: assigned version 0
 Whole bundle: 1 -> 1.1
The second component can replace the first one.
```

Some real-life issues

- Obtaining complete type information complicated
  - heterogeneous representation, placement
  - missing parts
- Language restrictions
  - subclass, not subtype, checks
    - ⇒ only ins/del differences allowed

Conclusion

- Subtype-based component substitution checks
  - possible by component type representation
  - practical through type difference classification
  - implemented for OSGi r4
- Issues (and non-issues)
  - as strong as type information allows
  - only subsequent revisions benefit from diff