Automated Versioning As a Mechanism for Component Software Consistency Guarantee

Jaroslav BAUML
Dept of Computer Science
UWB, Pilsen
Component Evolution

- Component updates reflect their evolution
- Application consistency during updates
  - One release of component
    - Dependencies to components
  - Releases during time
    - Dependencies to explicit releases of components (specified by version)
Versions

• Versions are one of the most important thing in component development...
  Do not believe?
Versions

• Versions are one of the most important things in component development...
Versions

• Versions are one of the most important thing in component development...
  ... because they handle (or describe) component evolution
• Unfortunately is also one of the less tracked
• Common versioning scheme:
  \textit{major.minor.micro\_textualAddons}
  \texttt{1.4.2\_alpha2}
Current use of version semantics

• Very often versioning only adhoc
• Marketing purpose
• Semantic purpose
  – All incompatible changes $\iff$ major $id$++.
  – All backward compatible $\iff$ minor $id$++.
  – No changes on interface $\iff$ micro $id$++.
Semantic Versioning

• No tooling, even no methodology to create semantic versioning.
• Specifications talk about “compatibility”, but it is too weak.
• To determine what is and what is not compatible, we can use structural subtyping.
Subtyping as key solution for semantic versioning

- Can use **Car** instead of **Vehicle**. **Car** is a subtype of ...
- **A** compatible with **B** $\iff$ **B** $\llhd$ **A**

  Difference values: **NON**, **MUT**, **SPE**, **GEN**
  - evaluate at leaves
  - *combine to root*:
    - $\text{spe} \oplus \text{gen} \to \text{mut}$
    - $\text{mut} \oplus X \to \text{mut}$
    - $\text{non} \oplus X \to X$
Subtyping Example

\[ R_{\text{old}} = \text{maj}_{\text{old}} \cdot \text{min}_{\text{old}} \cdot \text{mic}_{\text{old}} \]

\[ R_{\text{new}} = \text{maj}_{\text{new}} \cdot \text{min}_{\text{new}} \cdot \text{mic}_{\text{new}} \]
Subtyping Example (2)

• Two consecutive revisions, versions:
• The function *Difference* \((R_{\text{old}}, R_{\text{new}})\) is defined by the rules in the table.

<table>
<thead>
<tr>
<th>Diff((R_{\text{old}}, R_{\text{new}}))</th>
<th>(\text{maj}_{\text{old}})</th>
<th>(\text{min}_{\text{old}})</th>
<th>(\text{mic}_{\text{old}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>NON</td>
<td>(\text{maj}_{\text{new}})</td>
<td>(\text{min}_{\text{new}})</td>
<td>(\text{mic}_{\text{new}} + 1)</td>
</tr>
<tr>
<td>SPE</td>
<td>(\text{maj}_{\text{new}})</td>
<td>(\text{min}_{\text{new}} + 1)</td>
<td>0</td>
</tr>
<tr>
<td>GEN, MUT</td>
<td>(\text{maj}_{\text{new}} + 1)</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

OLD Logger 1.2.4, we found Mutation => NEW Logger‘s version = 2.0.0
Our Implementation for OSGi

• Simple meta-models
• Implementation of subtyping
• Java level
  – JavaTypes – general project to handle java type system at runtime
  – Bytecode, reflection, ...
• Bundle level
  – Metadata inspection
Validation – Real World Bundles

• To provide a basic validation of proposed principle, implemented tool was tested on bundles from Apache Felix archive.

<table>
<thead>
<tr>
<th>Bundle: org.apache.felix.fileinstall</th>
<th>Release</th>
<th>Diff()</th>
<th>Accordance</th>
<th>Correct version</th>
<th>Versions for Chain</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9.0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.9.0</td>
</tr>
<tr>
<td>0.9.2</td>
<td>SPE</td>
<td>No</td>
<td></td>
<td>0.10.0</td>
<td>0.10.0</td>
</tr>
<tr>
<td>1.0.0</td>
<td>GEN</td>
<td>Yes</td>
<td></td>
<td>1.0.0</td>
<td>1.0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bundle: org.apache.felix.framework</th>
<th>Release</th>
<th>Diff()</th>
<th>Accordance</th>
<th>Correct version</th>
<th>Versions for Chain</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0.1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1.0.1</td>
</tr>
<tr>
<td>1.0.3</td>
<td>NON</td>
<td>Yes</td>
<td></td>
<td>1.0.2</td>
<td>1.0.2</td>
</tr>
<tr>
<td>1.0.4</td>
<td>NON</td>
<td>Yes</td>
<td></td>
<td>1.0.4</td>
<td>1.0.3</td>
</tr>
<tr>
<td>1.2.0</td>
<td>MUT</td>
<td>No</td>
<td></td>
<td>2.0.0</td>
<td>2.0.0</td>
</tr>
<tr>
<td>1.2.1</td>
<td>NON</td>
<td>Yes</td>
<td></td>
<td>1.2.1</td>
<td>2.0.1</td>
</tr>
<tr>
<td>1.2.2</td>
<td>NON</td>
<td>Yes</td>
<td></td>
<td>1.2.2</td>
<td>2.0.2</td>
</tr>
<tr>
<td>1.4.0</td>
<td>SPE</td>
<td>Yes</td>
<td></td>
<td>1.3.0</td>
<td>2.1.0</td>
</tr>
<tr>
<td>1.4.1</td>
<td>SPE</td>
<td>No</td>
<td></td>
<td>1.5.0</td>
<td>2.2.0</td>
</tr>
<tr>
<td>1.6.0</td>
<td>SPE</td>
<td>Yes</td>
<td></td>
<td>1.5.0</td>
<td>2.3.0</td>
</tr>
<tr>
<td>1.8.0</td>
<td>NON</td>
<td>No</td>
<td></td>
<td>1.6.1</td>
<td>2.3.1</td>
</tr>
</tbody>
</table>
Conclusion

• Our method
  – allows you to ensure component consistency
  – possible by component type representation
  – as strong as type information allows (semantic changes, dynamic features)

• “No matter how sophisticated a (versioning) scheme is, it is useless if it is not used by all components universally.”