



OSGi Framework Modularity Improvements

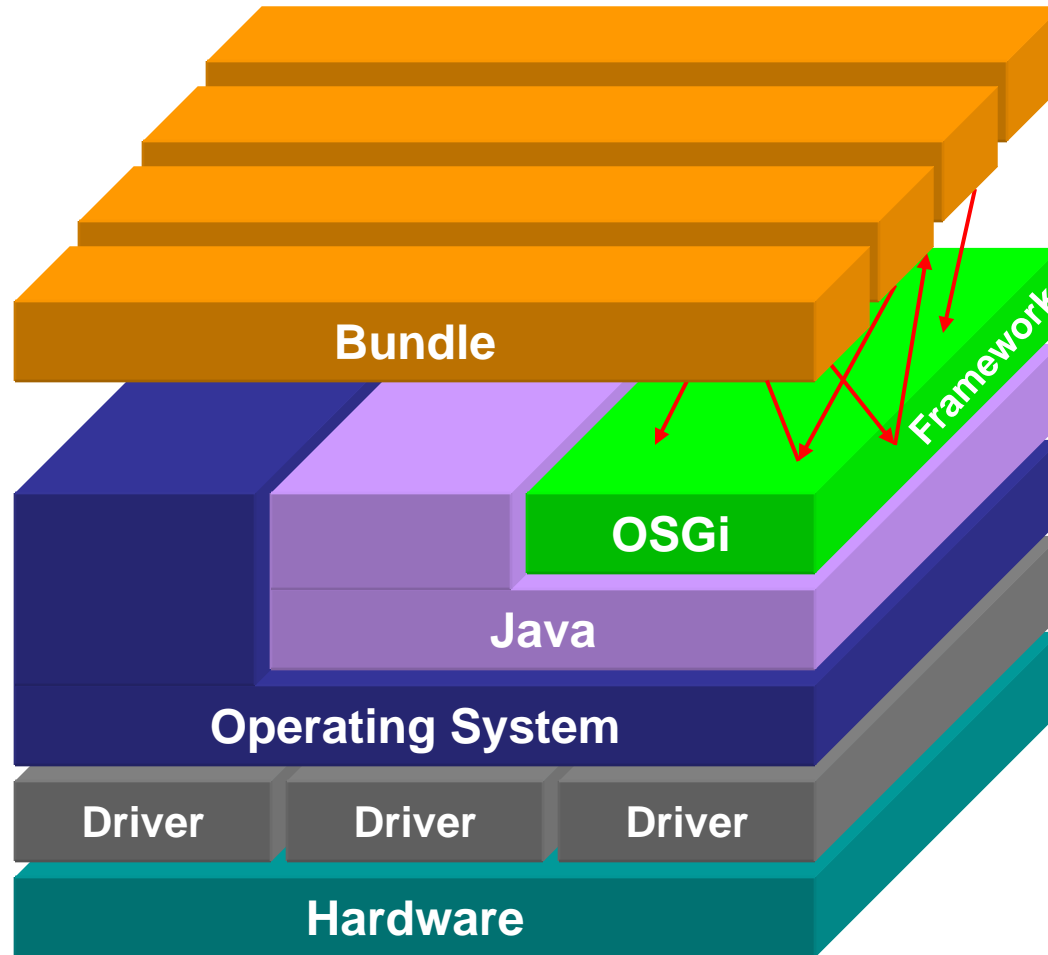
Glyn Normington, IBM UK Limited



Contents

- OSGi Framework
- What is Modularity?
- Requirements and Use Cases
- Evolution: OSGi R3, RFC 70, RFC 79
- Simple Examples

OSGi Framework



What is Modularity?

- “(Desirable) property of a system, such that individual components can be examined, modified and maintained independently of the remainder of the system. Objective is that changes in one part of a system should not lead to unexpected behavior in other parts.”

www.maths.bath.ac.uk/~jap/MATH0015/glossary.html

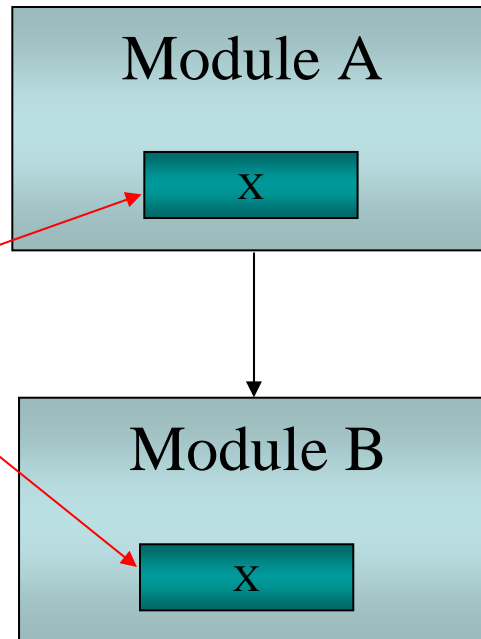
- For the purposes of this discussion we are focusing on the Java execution environment
- Class loading is the primary tool to enforce modularity
 - This provides for isolation between modules except for defined interactions
 - Customer class loaders not easy for application developers

Modularity Requirements

- Static analysis
 - e.g. reasoning about how a module will behave when combined with other modules
- Intra-module static optimisations
 - based on strictly-enforced module boundaries
 - e.g. a class private to a module and with no subclasses may be treated as final
- Existing code to be modularised without change
 - e.g. no need to change the package of a class

Use Case: Module Privacy

Both module A and module B need their own, distinct versions of a class named X.

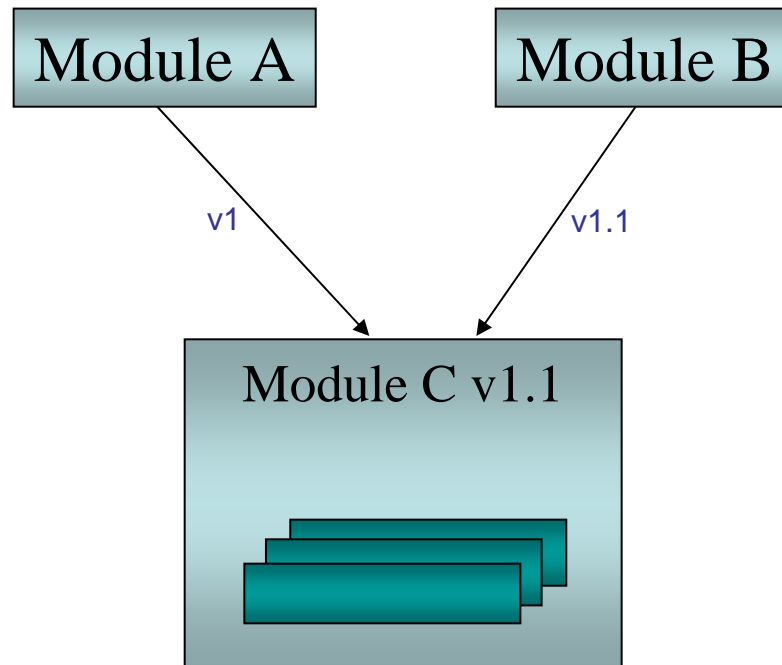


Key:

Module

Class

Use Case: Compatible Version Sharing



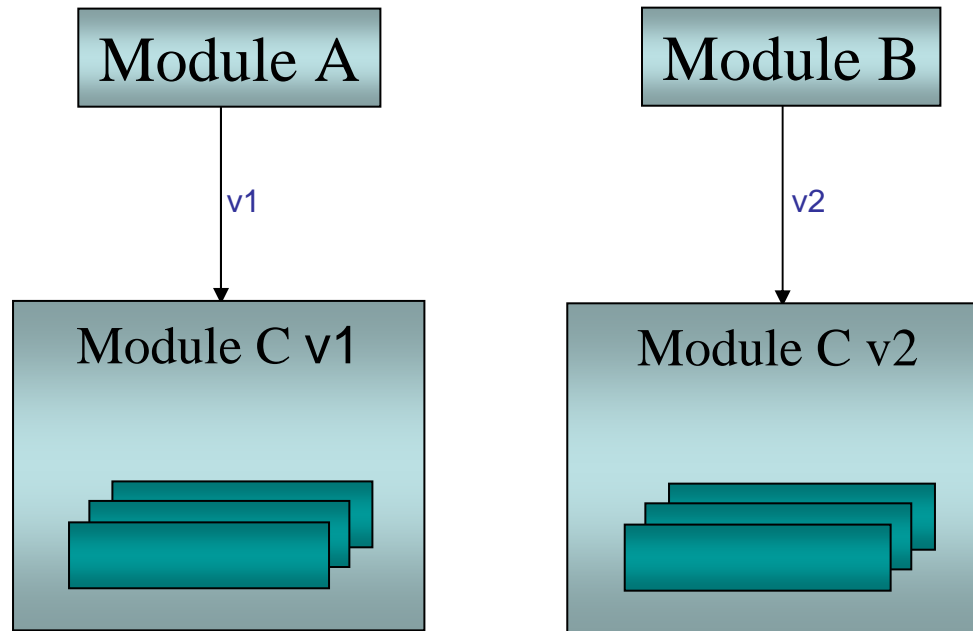
Note that a simple case of this scenario is where module A and module B require the **same** version of module C

Key:

Module

Class

Use Case: Incompatible Version Separation



Key:

Module

Class

OSGi Release 3

- Modularity features
 - Java package sharing between bundles
 - One bundle “exports”; other bundles “import”
 - Anonymous sharing
 - Single version of package may be shared
 - Simple version constraint (\geq)
- Generally suitable for embedded devices with newly developed bundles

RFC 70

- Used by Eclipse 3.0
 - RFC 70 is a post-R3 design effort to support modularity concepts of Eclipse plug-ins
 - Eclipse 3.0 includes an OSGi framework which implements OSGi R3 spec plus RFC 70 design
- New modularity features
 - Grouped Java package sharing between bundles
 - Multiple bundles “provide”; other bundles “require”
 - Named sharing
 - Multiple versions of a package may be shared
 - Version range constraints using interval notation e.g. [minv, maxv]
- Suitable for larger systems with legacy code and implementation dependencies

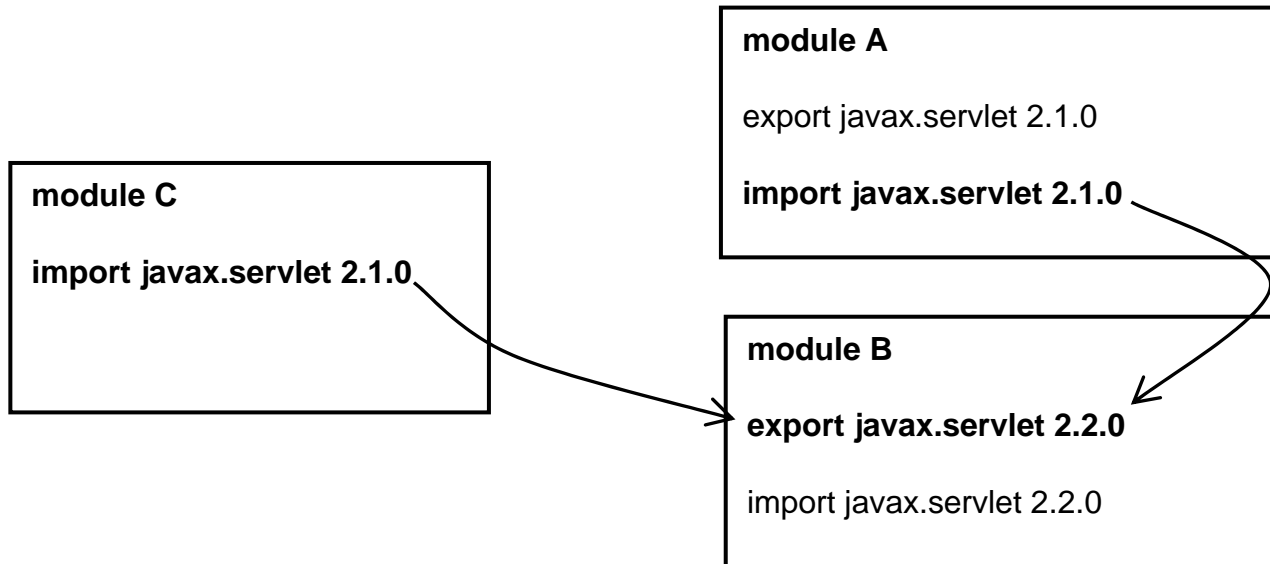
Release 3 + RFC 70 Sufficient?

- Opposite extremes of package sharing
 - R3: loose coupling to shared exported version
 - RFC 70: tight coupling to particular bundle
- Syntax/semantics not unified
- Some interesting intermediate cases are not addressed
 - e.g. loose coupling to matching exported version

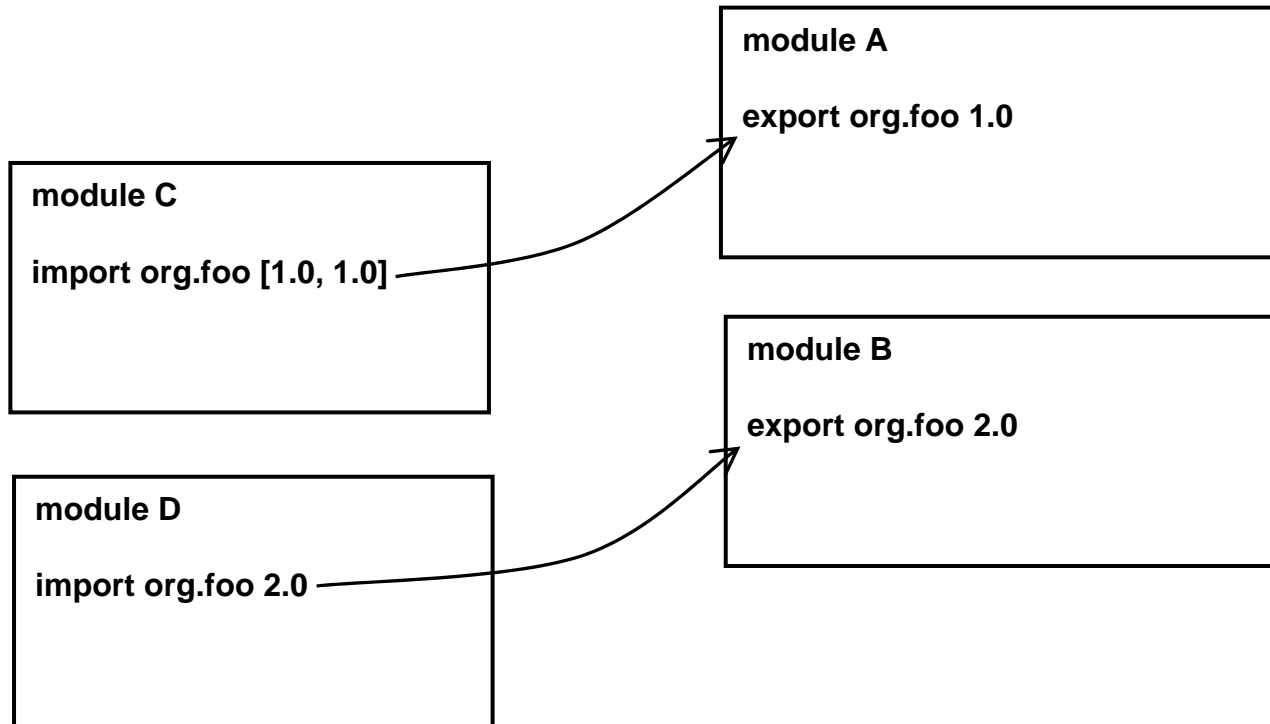
RFC 79 – Work in Progress

- Modularity features of RFC 79
 - Unified metadata encompassing R3 and RFC 70 needs as well as broader requirements
 - Targeted for OSGi Release 4
- Prototyping in Eclipse Technology Project

R3 Interface Sharing Example



Implementation Sharing Example



Attribute Matching Example

