New Modularity Features of the OSGi R4 Service Platform

Paris, France

Richard S. Hall
Motivation

Improving Upon Success
OSGi Success as a Modularity Framework

• OSGi framework is increasingly used as a modularity mechanism for Java
  – Provides logical and physical system structuring
    • Has benefits for development and deployment
  – Provides sophisticated dynamic module life-cycle management
    • Simplifies creation of dynamically extensible systems
      – Where system components can be added, removed, or rebound at run time while the system as a whole continues to function

• However, modularity was not the original goal...
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)
- Different types of modularity
  - **Logical**
    - Useful during development to decompose and/or structure the system
  - **Physical**
    - Useful after development to simplify deployment and maintenance
Why Care About Modularity?

• Simplifies the creation of large, complex systems
  – Improves robustness
  – Eases problem diagnosis
  – Enables splitting work among independent teams

• Simplifies the deployment and maintenance of systems

• Simplifies aspects of extensible and dynamic systems

• Java needs improvement in this area
  – Java currently lags .NET in support for modularity
  – OSGi specification deals with many of these issues and can fill that gap
Java & OSGi R3
Modularity Support & Limitations
Standard Java Modularity Mechanisms

- **Classes**
  - Provide logical static scoping via access modifiers (i.e., `public`, `protected`, `private`)

- **Packages**
  - Provide logical static scoping via `package privates`
  - Namespace mechanism, avoids name clashes

- **Java Archive (JAR) files**
  - Provide form of physical modularity
    - May contain applications, extensions, or services
    - May declare dependencies
    - May contain package version and sealing information

- **Class loaders**
  - Enable run-time code loading
  - Provide logical dynamic scoping
Standard Java Modularity Limitations (1)

- Limited scoping mechanisms
  - No module access modifier

- Simplistic version handling
  - Class path is first version found
  - JAR files assume backwards compatibility at best

- Implicit dependencies
  - Dependencies are implicit in class path ordering
  - JAR files add improvements for extensions, but cannot control visibility

- Split packages by default
  - Class path approach searches until it finds, which can lead to shadowing or mixing of versions
  - JAR files can provide sealing
Standard Java Modularity Limitations (2)

- **Low-level support for dynamics**
  - Class loaders are complicated to use

- **Unsophisticated consistency model**
  - Cuts across previous issues, it is difficult to ensure class space consistency

- **Missing module concept**
  - Classes too fine grained, packages too simplistic, class loaders too low level
  - JAR files are best candidates, but still inadequate
  - Modularity is a second-class concept as opposed to the .NET platform
    - In .NET, Assembly usage is enforced with explicit versioning rules and sharing occurs via the Global Assembly Cache
OSGi R3 Modularity

• Improves upon standard Java modularity
  – Defines *bundle*, a logical and physical modularity unit
    • Explicit dependencies
    • Explicit versioning
    • Internal class path concept
    • Code isolation
    • Packaging format
  – Defines dynamic bundle life cycle
    • Possible to install, update, and uninstall code at run time
    • Replaces low-level class loaders
OSGi R3 Modularity Issues (1)

- **Package sharing is only global**
  - Cannot have multiple shared versions
- **Simplistic versioning semantics**
  - Always backwards compatible
- **Not intended for sharing implementation packages**
  - Only for specification packages, which was why the version model is simple
- **Provider selection is always anonymous**
  - No way to influence selection
OSGi R3 Modularity Issues (2)

- Simplistic consistency model
  - Consistency model based on single in-use version
  - No way to declare dependencies among packages

- Coarse-grained package visibility rules
  - Classes in a package are either completely visible to everyone or hidden

- Module content is not extensible
  - All content of the logical module must be included in the physical module

- Package dependencies are not always appropriate
  - Package metadata is cumbersome in large, complex systems, tightly coupled subsystems and in less structured legacy systems
OSGi R4 Framework
Modularity Support for the Future
Modularity Requirements

- Backwards compatible with OSGi R3
- Defined in terms of Java packages
  - Well-defined concept in Java
  - Maps nicely to class loaders
- Explicitly defined boundaries
- Explicitly defined dependencies
- Support for versioning and multi-versions
- Flexible, must support
  - Small to large systems
  - Static to dynamic systems
Related Work

• Module mechanisms
  – Mechanisms for Secure Modular Programming in Java (L. Bauer et al – Princeton University)
  – Units: Cool Modules for HOT Languages (M. Flatt and M. Felleisen – Rice University)

• Component and extensible frameworks
  – EJB, Eclipse, NetBeans

• Microsoft .NET
  – Assemblies and Global Assembly Cache
OSGi R4 Modularity (1)

- Limitation: Package sharing is only global
• **Limitation:** Package sharing is only global

• **Multi-version support**
  
  – Possible to have more than one version of a shared package in memory at the same time
  
  – General change of philosophy to the prior OSGi specifications
  
  – Has deep impact on service aspects as well as modularity
    
    • For a given bundle, the service registry is implicitly partitioned according to the package versions visible to it
    
    • Impact on services not explored further in this presentation
OSGi R4 Modularity (2)

- Limitation: Simplistic versioning semantics
OSGi R4 Modularity (2)

- Limitation: Simplistic versioning semantics
- Import version ranges
  - Exporters still export a precise version, but importers may specify an open or closed version range
  -Eliminates existing backwards compatibility assumption
Limitation: Simplistic versioning semantics

Import version ranges

- Exporters still export a precise version, but importers may specify an open or closed version range
- Eliminates existing backwards compatibility assumption

Import-Package: foo; version="[1.0.0,1.5.0)"
• **Limitation: Simplistic versioning semantics**

• **Import version ranges**
  - Exporters still export a precise version, but importers may specify an open or closed version range
  - Eliminates existing backwards compatibility assumption

  ```
  Import-Package: foo; version="[1.0.0,1.5.0)"
  ```

• **Limitation: Not intended for sharing implementation packages**
• **Limitation:** Simplistic versioning semantics

• **Import version ranges**
  - Exporters still export a precise version, but importers may specify an open or closed version range
  - Eliminates existing backwards compatibility assumption

  ```
  Import-Package: foo; version="[1.0.0,1.5.0)"
  ```

• **Limitation:** Not intended for sharing implementation packages

• **Multi-version sharing and importing version ranges**
  make implementation package sharing possible
OSGi R4 Modularity (3)

- Limitation: Provider selection is always anonymous
OSGi R4 Modularity (3)

- **Limitation:** Provider selection is always anonymous

- **Arbitrary export/import attributes**
  - Exporters may attach arbitrary attributes to their exports, importers can match against these arbitrary attributes
    - Exporters may declare attributes as mandatory
      - Mandatory attributes provide simple means to limit package visibility
  - Importers influence package selection using arbitrary attribute matching
- **Limitation:** Provider selection is always anonymous.
- **Arbitrary export/import attributes**
  - Exporters may attach arbitrary attributes to their exports, importers can match against these arbitrary attributes
    - Exporters may declare attributes as mandatory
      - Mandatory attributes provide simple means to limit package visibility
    - Importers influence package selection using arbitrary attribute matching

```
Export-Package: foo;
    version="1.0.0";
    myattr="myvalue"
```

```
Export-Package: foo;
    version="1.0.0"
```
• **Limitation:** Provider selection is always anonymous

• **Arbitrary export/import attributes**
  - Exporters may attach arbitrary attributes to their exports, importers can match against these arbitrary attributes
    - Exporters may declare attributes as mandatory
      - Mandatory attributes provide simple means to limit package visibility
  - Importers influence package selection using arbitrary attribute matching

```
Import-Package: foo;
version="1.0.0";
myattr="myvalue"
```
OSGi R4 Modularity (3)

- **Limitation:** Provider selection is always anonymous

- **Arbitrary export/import attributes**
  - Exporters may attach arbitrary attributes to their exports, importers can match against these arbitrary attributes
    - Exporters may declare attributes as mandatory
      - Mandatory attributes provide simple means to limit package visibility
  - Importers influence package selection using arbitrary attribute matching

```
Import-Package: foo;
version="1.0.0";
myattr="myvalue"
```
• Limitation: Simplistic consistency model
OSGi R4 Modularity (4)

- **Limitation:** Simplistic consistency model

- **Sophisticated package consistency model**
  - Exporters may declare package “uses” dependencies
    - Exported packages express dependencies on imported or other exported packages, which constrain the resolve process
  - The framework must ensure that importers do not violate constraints implied by “uses” dependencies
OSGi R4 Modularity (4)

- **Limitation:** Simplistic consistency model
- **Sophisticated package consistency model**
  - Exporters may declare package “uses” dependencies
    - Exported packages express dependencies on imported or other exported packages, which constrain the resolve process
  - The framework must ensure that importers do not violate constraints implied by “uses” dependencies

Export-Package: foo

Export-Package: foo
• **Limitation:** Simplistic consistency model

• **Sophisticated package consistency model**
  – Exporters may declare package “uses” dependencies
    • Exported packages express dependencies on imported or other exported packages, which constrain the resolve process
  – The framework must ensure that importers do not violate constraints implied by “uses” dependencies

```
Import-Package: foo, bar
```
• **Limitation:** Simplistic consistency model
• **Sophisticated package consistency model**
  – Exporters may declare package “uses” dependencies
    • Exported packages express dependencies on imported or other exported packages, which constrain the resolve process
  – The framework must ensure that importers do not violate constraints implied by “uses” dependencies

```
Import-Package: foo
Export-Package: bar;
uses:="foo"
```
OSGi R4 Modularity (4)

- **Limitation:** Simplistic consistency model
- **Sophisticated package consistency model**
  - Exporters may declare package “uses” dependencies
    - Exported packages express dependencies on imported or other exported packages, which constrain the resolve process
  - The framework must ensure that importers do not violate constraints implied by “uses” dependencies

```
Import-Package: foo
Export-Package: bar;
uses:="foo"
```
OSGi R4 Modularity (4)

- **Limitation: Simplistic consistency model**
- **Sophisticated package consistency model**
  - Exporters may declare package “uses” dependencies
    - Exported packages express dependencies on imported or other exported packages, which constrain the resolve process
  - The framework must ensure that importers do not violate constraints implied by “uses” dependencies

```
Import-Package: foo
Export-Package: bar;
uses:="foo"
```
• **Limitation:** Simplistic consistency model

• **Sophisticated package consistency model**
  – Exporters may declare package “uses” dependencies
    • Exported packages express dependencies on imported or other exported packages, which constrain the resolve process
  – The framework must ensure that importers do not violate constraints implied by “uses” dependencies

```
Import-Package: foo
Export-Package: bar;
uses:="foo"
```
OSGi R4 Modularity (4)

- **Limitation:** Simplistic consistency model
- **Sophisticated package consistency model**
  - Exporters may declare package “uses” dependencies
    - Exported packages express dependencies on imported or other exported packages, which constrain the resolve process
  - The framework must ensure that importers do not violate constraints implied by “uses” dependencies

```
Import-Package: foo
Export-Package: bar;
uses:="foo"
```
Limitation: Coarse-grained package visibility rules
OSGi R4 Modularity (5)

- Limitation: **Coarse-grained package visibility rules**
- Package filtering
  - Exporters may declare that certain classes are included/excluded from the exported package
- **Limitation:** Coarse-grained package visibility rules

- **Package filtering**
  - Exporters may declare that certain classes are included/excluded from the exported package

```
Export-Package: foo;
exclude:="*Impl",
foo; friend="yes";
mandatory:="friend"
```
Limitation: Coarse-grained package visibility rules

Package filtering

- Exporters may declare that certain classes are included/excluded from the exported package
**Limitation:** Coarse-grained package visibility rules

**Package filtering**

- Exporters may declare that certain classes are included/excluded from the exported package

```
Import-Package: foo

B: [foo
foo
(exclude:="*Impl")

A: [foo

C: foo

foo
friend="yes"
(include:="*")
```

```
Import-Package: foo;

friend="yes"
```
Limitation: **Coarse-grained package visibility rules**

- **Package filtering**
  - Exporters may declare that certain classes are included/excluded from the exported package

```
Import-Package: foo
```

```
foo
(exclude:="*Impl")
```

```
foo
friend="yes"
(include:="*")
```

```
Import-Package: foo;
friend="yes"
```
OSGi R4 Modularity (6)

- Limitation: Module content is not extensible
Limitation: Module content is not extensible

Bundle fragments

- A special bundle that attaches to a host bundle and uses the same class loader
  - Conceptually becomes part of the host bundle
OSGi R4 Modularity (6)

- **Limitation:** Module content is not extensible
- **Bundle fragments**
  - A special bundle that attaches to a host bundle and uses the same class loader
  - Conceptually becomes part of the host bundle

Fragment-Host: B
Export-Package: foo
Import-Package: baz

Bundle-SymbolicName: B
Export-Package: bar
Import-Package: woz
OSGi R4 Modularity (6)

- **Limitation:** Module content is not extensible
- **Bundle fragments**
  - A special bundle that attaches to a host bundle and uses the same class loader
    - Conceptually becomes part of the host bundle
Limitation: Module content is not extensible

Bundle fragments

- A special bundle that attaches to a host bundle and uses the same class loader
  - Conceptually becomes part of the host bundle
- **Limitation:** Module content is not extensible
- **Bundle fragments**
  - A special bundle that attaches to a host bundle and uses the same class loader
    - Conceptually becomes part of the host bundle
OSGi R4 Modularity (6)

- **Limitation:** Module content is not extensible
- **Bundle fragments**
  - A special bundle that attaches to a host bundle and uses the same class loader
    - Conceptually becomes part of the host bundle
Limitation: Package dependencies are not always appropriate
• **Limitation:** Package dependencies are not always appropriate

• **Bundle dependencies**
  – Import everything that another, specific bundle exports
  – Allows re-exporting
OSGi R4 Modularity (7)

• **Limitation:** Package dependencies are not always appropriate.

• Bundle dependencies
  – Import everything that another, specific bundle exports
  – Allows re-exporting

---

**Bundle-SymbolicName:** A  
**Export-Package:** bar, foo  
**Require-Bundle:** A  
**Export-Package:** bar
OSGi R4 Modularity (7)

- **Limitation:** Package dependencies are not always appropriate.
- **Bundle dependencies**
  - Import everything that another, specific bundle exports
  - Allows re-exporting

**Diagram:**

```
Bundle-SymbolicName: A
Export-Package: bar, foo

Require-Bundle: A
Export-Package: bar
```
Example OSGi R4 Bundle Manifest

Bundle-ManifestVersion: 2
Bundle-SymbolicName: org.foo.simplebundle
Bundle-Activator: org.foo.Activator
Bundle-ClassPath: .,org/foo/embedded.jar
Bundle-NativeCode:
  libfoo.so; osname=Linux; processor=x86,
  foo.dll; osname=Windows 98; processor=x86
Import-Package:
  javax.servlet; version="[2.0.0,2.4.0)";
    resolution:="optional"
Export-Package:
  org.foo.service; version=1.1; vendor="org.foo";
    exclude:="*Impl",
  org.foo.service.bar; version=1.1;
    uses:="org.foo.service"
Challenges

- Manage the complexity
  - Maintain conceptual integrity
  - Keep the simple cases simple
  - Complexity should only be visible when it is required
  - Avoid bloat to support small devices
Challenges

- Manage the complexity
  - Maintain conceptual integrity
  - Keep the simple cases simple
  - Complexity should only be visible when it is required
  - Avoid bloat to support small devices

The “good news” is that these changes generally only affect the dependency resolving algorithm.
Conclusions

• Java needs improved modularity support
  – We need to stop re-inventing the wheel
  – Improve application structure
  – Simplify deployment and management especially in technological areas where deployment is inherent
    • e.g., component orientation, extensible systems, and service orientation (to some degree)

• OSGi R1/R2/R3 were all steps in the right direction

• OSGi R4 goes even further in providing sophisticated Java modularity
  – OSGi technology is cited in JSR 277, an initiative by Sun to define a module system for Java, whose expert group includes OSGi members
Questions