Modular Applications and the Lookup API

Tim Boudreau
Senior Staff Engineer
Sun Microsystems
The Need for Modular Applications

- Applications get more complex
- Assembled from pieces
- Developed by distributed teams
- Components have complex dependencies
- Good architecture
  - Know your dependencies
  - Manage your dependencies
The Entropy of Software

- Version 1.0 is cleanly designed...
The Entropy of Software

• Version 1.1...a few expedient hacks...we'll clean those up in 2.0
The Entropy of Software

- Version 2.0...oops...but...it works!
The Entropy of Software

- Version 3.0...Help! Whenever I fix one bug, I create two more!
● Version 4.0 is cleanly designed. It's a complete rewrite. It was a year late, but it works...
The Entropy of Software

- Version 4.1...does this look familiar?....
The Entropy of Software

- TO BE CONTINUED....
Types of Library

• Simple library – one impl, put it on classpath and use
• Reference Impl + Vendor Impl – You trust that the Vendor impl conforms to the spec
• Modular Library – the API is separate from the implementation
  - Multiple implementations possible
  - Spec conformance is enforced by design
  - API must find its implementation
  - You need a registry of things
Modular Applications

- Discover their components at runtime
- May add/remove/reload components at runtime
- Must satisfy dependencies between components
- Have API contracts between components
- Run inside a runtime container
What is a NetBeans Module

• It is just a JAR file – no magic
  – Has some special manifest entries to describe it to NetBeans
  – Editable in the Project Properties dialog for module projects

• Distributed in an NBM file
  – Basically a signed JAR file
  – Contains metadata about the module
  – May contain 3rd party JARs or anything else that needs to be on the system
NetBeans Module Manifest

Manifest-Version: 1.0
Ant-Version: Apache Ant 1.7.0
Created-By: 1.5.0_14-b03 (Sun Microsystems Inc.)
OpenIDE-Module-Public-Packages: -
OpenIDE-Module-Module-Dependencies: org.netbeans.api.java/1, ...
OpenIDE-Module-Java-Dependencies: Java > 1.5
OpenIDE-Module-Build-Version: 200804211638
OpenIDE-Module-Specification-Version: 2.12.0.4.1.1.6
OpenIDE-Module: org.netbeans.modules.java.editor/1
OpenIDE-Module-Implementation-Version: 4
OpenIDE-Module-Localizing-Bundle:
  org/netbeans/modules/java/editor/Bundle.properties
OpenIDE-Module-Install:
  org/netbeans/modules/java/editor/JavaEditorModule.class
OpenIDE-Module-Layer:
  org/netbeans/modules/java/editor/resources/layer.xml
OpenIDE-Module-Requires: org.openide.modules.ModuleFormat1
AutoUpdate-Show-In-Client: false
Modular Runtime Containers Must

• Ensure dependencies are satisfied
  – Including requiring > version n of a module
• Not allow illegal dependencies
• Allow legal dependencies
• Instantiate components of the system at runtime
• Provide service registration/discovery facility
What is a NetBeans Module

- It is just a JAR file – no magic
  - Has some special manifest entries to describe it to NetBeans

- Distributed in an NBM file
  - Basically a signed JAR file
  - Contains metadata about the module
  - May contain 3rd party JARs or anything else that needs to be on the system
Enforcing Module Dependencies

Module A

com.myapp.mymodule

Has no public API

Module B

com.myapp.api

com.myapp.impl

A declares a dependency on B
B allows has some "public packages" (API)
A can see classes in com.myapp.api
A can **not** see classes in com.myapp.impl

public API package
Use an Existing Runtime Container

Rest In Peace, Home-made Frameworks 1995-2005
Module Dependencies

- Core Runtime
  - Text API
  - Editor
    - Java Editor
    - Java Parser (javac)
  - Projects API
    - Apache Ant Project Support
Provides/Requires Tokens

- API can be in one module, implementation in another
- API module can include a requires token in its manifest
  
  OpenIDE-Module-Requires: Spellchecker

- Implementation module includes a provides token in its manifest
  
  OpenIDE-Module-Provides: Spellchecker

- Modules needing the API only install if requirement is satisfied
Modular Libraries and Discovery

org.netbeans.modules.text
**Provides:** EditorImpl
**Requires:** Spellchecker

org.netbeans.spellimpl
**Provides:** Spellchecker

org.netbeans.api.spellchecker

RUNTIME ENGINE
Discovery and Dependencies

So how will the SpellChecker API find its implementation?
The Java Extension Mechanism

- In JDK since 1.3
- Easy with JDK 6's ServiceLoader.load()
- Declarative registration
  - No startup penalty
- Plain-text file in META-INF/services
  - Name is interface
  - Content is FQN of implementation
Other Solutions

**Global static singleton – why that's bad**
- Why that's bad:
  - Can never be garbage collected
  - Locked in to one implementation

**Setter injection – why that's bad:**
- Can be changed later by foreign code
- A modular application may contain modules the original author did not write
- Introduces state – threading and synchronization issues
- “Push” model where we should be using a “pull” model

**String-based registry (JNDI, etc.) - why that's bad:**
- Not type-safe
Lookup – NetBeans Solution

- Small, NetBeans independent library
  - Part of NetBeans org-openide-util.jar
    - `org.openide.util.Lookup`

- Works with any version of Java (unlike JDK's ServiceLoader)
- A Lookup is dynamic
  - Can fire changes

- A Lookup is instantiable
  - You can make one and use it

- Lookups are composable
  - ProxyLookup can combine and switch between other lookups and fire changes
A Lookup is a place

• A space objects swim into and out of
• You can observe when specific types of objects appear and disappear
• You can get a collection all of the instances of a type in a Lookup
The Default Lookup – A global registry

• Global Lookup Patterns
  – Pseudo-singletons:
    ```java
    StatusDisplayer x = Lookup.getDefault().lookup(
      StatusDisplayer.class);
    ```

  • Better memory management: The singleton can be garbage collected if nothing references it

  – Global services
    ```java
    Collection <? extends SomeClass> c =
    Lookup.getDefault().lookupAll( ProjectFactor
    y.class );
    ```
Lookup: Service discovery and more
Can Contain >1 instance of a type

- It's not just for singletons
- Requesting multiple objects is easy:
  
  ```java
  Collection <? extends A> c = Lookup.getDefault().lookupAll(A.class);
  ```
Lookup.Result<A> result = Lookup.getDefault().lookup(A.class);
Tracking Changes in a Lookup
Listening To A Lookup.Result

• Why do that?
  – Default Lookup:
    • Detect when a module is uninstalled/installed that provides something you are interested in
  – Some object that owns a lookup
    • Detect when the set of its “capabilities” change
Listening for Changes

```java
Lookup.Result<SomeClass> r = someLookup.lookupResult ( SomeClass.class );

r.addLookupListener ( new LookupListener() {
    public void resultChanged (LookupEvent e) {
        //handler code here
    }
});
```
So...What's So Special About This?

? 

What if objects had Lookups?
What if Lookups could proxy each other?
Example: NetBeans Project API

- Associates a directory on disk with a Lookup
- Defines interfaces that may be in that Lookup

```java
public interface Project extends Lookup.Provider {
    FileObject getProjectDirectory();
    Lookup getLookup();
}
```
Each main window tab has its own Lookup
- Some tabs show Nodes, which also have Lookups, and proxy the selected Node's Lookup

A utility Lookup proxies the Lookup of whatever window tab has focus
- What is “to proxy”?

Lookup lkp = Utilities.actionsGlobalContext();
Example: Selection in NetBeans

Demo
Creating Your Own Lookup – when?

• When do you want to do this? Common cases:
  – You are implementing a Project
    • The Lookup provides objects that let code interact with the project
  – You are writing a TopComponent (logical window)
    • The Lookup provides its selection
  – You are writing a Node
    • The Node's Lookup contents determine what actions will be enabled, what is shown in the Navigator, etc.
  – You are creating an API that other modules can inject objects into
    • Your API classes can be final but still be extensible
Creating Your Own Lookup - How?

• A Lookup that never changes
  - `org.openide.util.lookup.Lookups`
    • A utility class that provides some convenient Lookup implementations

• You set the contents once and it stays this way forever

```java
Lookup lkp = Lookups.fixed ( obj1, obj2, obj3 );
Lookup lkp = Lookups.singleton( onlyObject );
```
Creating Your Own Lookup - How?

- AbstractLookup – lookup subclass
  - `org.openide.util.lookup.AbstractLookup`
  - Driven by an InstanceContent object
  - You can add/remove/set the contents on the fly
    - Appropriate changes will be fired to listeners

```java
InstanceContent content = new InstanceContent();
Lookup lkp = new AbstractLookup(content);
content.set(obj1, obj2, obj3);
content.remove(obj3);
```
Creating Your Own Lookup - How?

- **ProxyLookup**
  - Merge multiple lookups together
    - A lookup that *proxies* a bunch of other lookups
  - Can change which lookups are merged together on the fly
    - And appropriate events will be fired

```java
Lookup lkp = new ProxyLookup ( otherLookup1, otherLookup2, otherLookup3 );
```
ProxyLookup

Lookup

BarImpl

FooImpl

BazImpl

BImpl

AImpl

FooImpl
ProxyLookup
Useful Utility Implementations

- AbstractLookup + InstanceContent
  - Lookup whose contents you can manage
- Lookups.singleton( Object ) - one item Lookup
- Lookups.fixed( Object... ) - unchanging Lookup
- Lookups.exclude ( Lookup, Class... );
- ProxyLookup ( Lookup... otherLookups ) - compose multiple lookups
Named Global Lookups

• New in NetBeans 6
• Many “global” lookups
  - `Lookup myOwnRegistry = Lookups.forPath("my/registry/path");`
• Standalone
  - META-INF/namedservices/my/registry/path
• Integrated with System File System
Conclusion

- Lookup is used pervasively throughout NetBeans APIs
- It is used for
  - Declaratively registered global services
    - Instantiation on demand – reduce startup time
    - Separation of API and implementation
      - One module can provide an API
      - Another module provides the implementation
  - Selection context – action enablement & more
  - Simplifying general-purpose APIs (such as Project)
- It is one of the most important APIs to learn
References

• Lookup Javadoc:
  http://bits.netbeans.org/dev/javadoc/org-openide-util/org/openide/util/Lookup.html

• Get the library
  $NB_HOME/platform8/lib/org-openide-util.jar

• Article
  - http://openide.netbeans.org/lookup/

• FAQ
  http://wiki.java.net/bin/view/Netbeans/NetBeansDeveloperFAQ
Questions & Answers

Q&A