Modular Applications and the Lookup API

Geertjan Wielenga
NetBeans Team
The Need for Modular Applications
The Need for Modular Applications
The Need for Modular Applications
The Need for Modular Applications

- Applications get more complex
- Assembled from pieces
- Developed by distributed teams
- Components have complex dependencies

Healthy architecture:
- Know your dependencies
- Manage your dependencies
• 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....
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 Does a Runtime Container Do?

• Application lifecycle
  > Starts and exits your application
  > Modules are installed and uninstalled

• Module discovery and management

• Classloading and code isolation

• Service registration/discovery facility
What is the NetBeans Runtime Container?
Demo 1: NetBeans Runtime Container
NetBeans Runtime Container

• 5 NetBeans Platform modules

• These 5 together compile & run, nothing more required.

• **Only** these are required by all NetBeans Platform applications

• So... by default, NetBeans Platform applications are not all client applications

• Modular server applications on the NetBeans Platform?
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 3\textsuperscript{rd} party JARs or anything else that needs to be on the system
Demo 2: Creating a NetBeans Module
Demo 3: Distributing a NetBeans Module
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
NetBeans Module Manifest

**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
Runtime Container Tasks

- Ensure that dependencies are satisfied
  > Including requiring > version \( n \) of a module

- Not allow illegal dependencies

- Allow legal dependencies

- Instantiate components of the system at runtime
Enforcing Module Dependencies

Module A
- com.myapp.mymodule
- 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

Module B
- com.myapp.api
- com.myapp.impl

Has no public API
Use an Existing Runtime Container

Rest In Peace, Home-made Frameworks
1995-2005
Separate Implementations from API

- API can be in one module, implementation in another
- Modules needing the API only install if dependencies are satisfied
Discovery and Dependencies

- So how will the SpellChecker API find its implementation?
Discovery and Dependencies

- So how will the SpellChecker API find its implementation?
- In general, how do NetBeans modules find each other within the application?
The Java Extension Mechanism (almost it)

- 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
Demo 5: ServiceLoader

• Interface

```java
public interface TextFilter {
    String process(String s);
}
```

• Implementation

```java
public class UpperCaseFilter implements TextFilter{
    public String process(String s) {
        return s.toUpperCase();
    }
}
```
Demo 5: ServiceLoader

- Register the Implementation
- Load the Interface

```java
String s = textArea.getText();

ServiceLoader<TextFilter> filters = ServiceLoader.load(TextFilter.class);
for (TextFilter textFilter : filters) {
    if (filters != null) {
        s = textFilter.process(s);
    }
}

textArea.setText(s);
```
String s = textarea.getText();

Collection<? extends TextFilter> filters = Lookup.getDefault().lookupAll(TextFilter.class);

for (TextFilter textFilter : filters) {
    if (filters != null) {
        s = textFilter.process(s);
    }
}

textArea.setText(s);
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 object appear and disappear
- You can get a collection all of the instances of a type in a Lookup
So...What's So Special About This?

What if objects had Lookups?
What if Lookups could proxy each other?
Objects Have Lookups Too!

- TopComponent
- Node
- DataObject
Demo 7: TopComponent Lookup
Demo: TopComponent Lookup

SaveAction

s == null ?

yes

disable action

no

enable action

on action invocation:
call s.save()

give me a SaveCookie

interface SaveCookie {
    void save();
}

Editor
private InstanceContent content;

...

...

content = new InstanceContent();
associateLookup(new AbstractLookup(content));

...

...

content.add(s);
private Lookup.Result result;

...  
...
...
result = 
  Utilities.actionsGlobalContext().lookupResult(String .class);
result.addLookupListener(new LookupListener() {
  @Override
  public void resultChanged(LookupEvent e) {
    textArea2.setText(result.allInstances().toString());
  }
});
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
  > Context sensitivity, e.g., action enablement

• It is the most important APIs to play with!
The Need for Modular Applications
Revision of New Concepts

- Application Entropy
- Modularity
- Dependency Management
- Runtime Container
- NetBeans Module
- NBM File
- API vs Implementation
- META-INF/services
- ServiceLoader vs. Lookup
- Listening to the Lookup
- Context sensitivity via Lookup
Questions & Answers