MJ: A Rational Module System for Java and its Applications

John Corwin, David Bacon, David Grove, Chet Murthy
IBM
OOPSLA 2003
What is in JAVA?

- Class
  - OO encapsulation
- Package
  - Static Namespace
- Classloader
  - Dynamic Namespace
Review: Package in Java

- Package groups related classes and interfaces and provides access protection and namespace management statically.
- Package offers an access level between class and subclass, world.
- "world" can only access public classes declared in a package.
Review: Java Classloader

- Classloader is responsible for converting Java bytecode into Java class files
- Classloaders are organized in a hierarchy. The root of all class loaders is called the system classloader.
- Policies are defined for each classloader regarding whether a class is allowed to load, access or not.
  - Dynamic Namespace
- The child classloader has to consult the parent to load a class.
- Help manage unknown code, specifically designed for security reason
What is missing in JAVA?

★ What they need
  - Flexible component interface

★ Class
  - Too fine-grained to define a component

★ Package
  - No nested component
  - Expose same interface to outside world

★ Classloader
  - Policy hard-coded in the program
  - No static checking
What they do and how?

- **Motivation:** remove classloader from codes (efficiency?)
- **MJ module system**
  - Add-on to java
  - Module enabled Javac
  - Static checking
- **How?**
  - Classloader
Problem: Abstract Datatype

Need: an object with implementation class \( C \), providing only an abstract interface \( A \) to some clients, while providing a more extensive interface to others.

- e.g. There is no safe backdoor for \( \text{java.lang.String} \)
  
  ```java
  public class String{
      private char value[];
      public void getChars (int srcBegin, int srcEnd, char[] dst, int dstBegin)
  }
  
  Whenever you need the character array of a string, you have to copy it out first
Problem: Multiple Versions of the Same Class

- XML parser, XSL processor
  - XML APIs are changing very rapidly
- Different ORBs
  - IONA Orbix, Visibroker
- But same class name, each ORB has its own different ORB will have its own name space (classloader)
More Problems:

* Classpath management for programmers
* Missed optimization opportunities
* All mentioned problems are not related to security
A Module System For Java: A Programmer’s View

A module’s metadata consists of the following information:

- Which classes this module provides, and where these classes are archived
- Which other modules does this module depend on, and which classes do we require from each of these modules
- Access Control for this module’s provided classes
- Initialization Code
Sample Module Definition (.jm file)

provides "catalina.jar";
import * from xerces;
import * from bootstrap;
import com.sun.tools.* from tools;

hide * in *;
export org.apache.catalina.* to
    webapp;
export org.apache.catalina.servlets
    to servlets;

forbid org.apache.catalina.* in *;

module catalina {
    public static void load() { .... }
    public static void main(String[]
        args) { ... }
}
Sample Module Definition: Module Name

provides "catalina.jar";
import * from xerces;
import * from bootstrap;
import com.sun.tools.* from tools;

hide * in *;
export org.apache.catalina.* to webapp;
export org.apache.catalina.servlets to servlets;

forbid org.apache.catalina.* in *;

module catalina {
    public static void load() { ... }
    public static void main(String[] args) { ... }
}
Sample Module Definition: Java Resources

provides "catalina.jar";
import * from xerces;
import * from bootstrap;
import com.sun.tools.* from tools;

hide * in *;
export org.apache.catalina.* to webapp;
export org.apache.catalina.servlets to servlets;

forbid org.apache.catalina.* in *;

module catalina {
    public static void load() { .... }
    public static void main(String[] args) { ... }
}
Sample Module Definition: Import

provides "catalina.jar";
import * from xerces;
import * from bootstrap;
import com.sun.tools.* from tools;

hide * in *;
export org.apache.catalina.* to webapp;
export org.apache.catalina.servlets to servlets;

forbid org.apache.catalina.* in *;

module catalina {
    public static void load() { .... }
    public static void main(String[] args) { ... }
}
Sample Module Definition: Export

provides "catalina.jar";
import * from xerces;
import * from bootstrap;
import com.sun.tools.* from tools;

hide * in *;
export org.apache.catalina.* to webapp;
export org.apache.catalina.servlets to servlets;

forbid org.apache.catalina.* in *;

module catalina {
    public static void load() { .... }
    public static void main(String[] args) { ... }
}
Sample Module Definition: Forbid

provides "catalina.jar";
import * from xerces;
import * from bootstrap;
import com.sun.tools.* from tools;

hide * in *
export org.apache.catalina.* to webapp;
export org.apache.catalina.servlets to servlets;

forbid org.apache.catalina.* in *;

* forbid statements prohibit any importer from implementing a set of classes and from importing such classes from any other module

* Classloader has the corresponding concept
Sample Module Definition: load/unload/main

provides “catalina.jar”;
import * from xerces;
import * from bootstrap;
import com.sun.tools.* from tools;

hide * in *;
export org.apache.catalina.* to webapp;
export org.apache.catalina.servlets to servlets;

forbid org.apache.catalina.* in *;

module catalina {
    public static void load() { .... }
    public static void main(String[] args) { ... }
}
Other Module Declarations

- seal and unseal declarations to control ability to subclass
- dynamic export and hide
Modularized Apache Tomcat

- No module division, just use libraries and jars as modules
- Basically it is to remove classloader from the code
Classloaders in Tomcat

* Thread Context Classloader
  - The request to the parent classloader breaks down
* Ad-hoc modularity
* Dynamic loading of servlets and JSPs
  - Dynamic loading is necessary
* Some unnecessary ones
Module relationships in Apache Tomcat
Servlet Microbenchmark

* Implemented java.lang.StringInternals class
  ```java
class StringInternals {
    public static final char[] getValue(String s) {
        return s.value;
    }
}
```

* MJ makes StringInternals available only to trusted code

* Modified java.io.OutputStreamWriter to exploit StringInternals

* Achieved 30% speedup in Servlet microbenchmark on JDK 1.3.1
  - Run on modularized tomcat
  - Servlet generates an HTML multiplication table
Related Work

* **Component Systems**
  - Microsoft COM object model
  - OSGi bundles; Eclipse plugins
  - Access control is new?

* **Module Systems**
  - Units [Findler & Flatt]; ML functors [MacQueen]
  - Ada, Modula-2, Modula-3, Haskell

* **Java Extensions**
  - Jiauzzi [McDirmid et al]; Component J [Seco & Caires]; [Bauer et al]
  - JavaMod [Ancona & Zucca]; Multi-Java [Clifton et al]