Pluggable Reflection: Decoupling Meta-Interface and Implementation

David H. Lorenz
Northeastern University
College of Computer & Information Science
Boston, MA 02115
`lorenz@ccs.neu.edu`

Joint Work With
John Vlissides
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
`vlis@us.ibm.com`

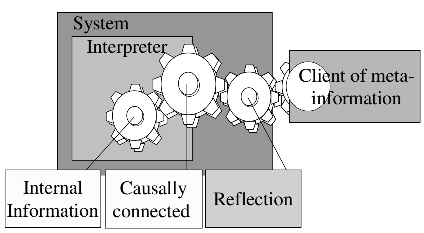
Research

Meta-
Decoupling Interface and Implementation

- **Reflection**
  - **Depends on implementation**
  - **Second-class citizen in current programming models**

What's the problem with Reflection?

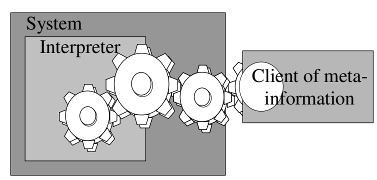
- **Strong coupling between reflection interface and implementation**
  - **Code that uses reflection is assumed to access the reflective representation---they are one and the same**
  - **java.lang.reflect.* are final classes, not interfaces!**



Motivation

☞ **We seek decoupling of Reflection and clients of meta-information.**

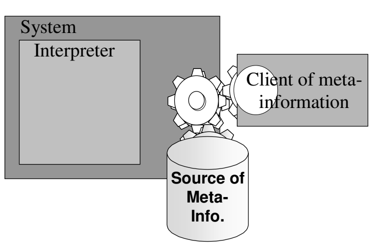
- **The need arises as soon as you have more than one source of meta-information**



Motivation

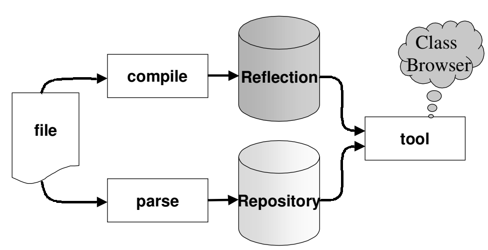
☞ **We seek decoupling of Reflection and clients of meta-information.**

- **The need arises as soon as you have more than one source of meta-information**



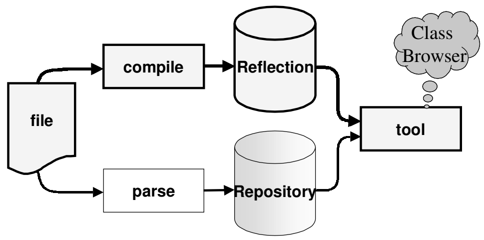
Reflection-Repository Duality

- **Overlap between reflective facilities and source code repository**
  - **Manage information about classes and their relationships**
  - **Aid in the software development process**



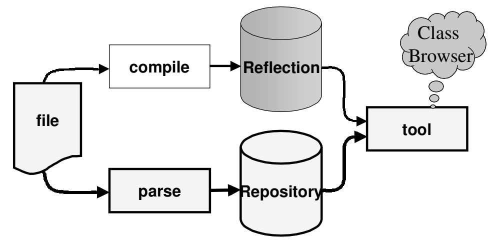
Reflection-Repository Duality

- Overlap between reflective facilities and source code repository
  - Manage information about classes and their relationships
  - Aid in the software development process



Reflection-Repository Duality

- Overlap between reflective facilities and source code repository
  - Manage information about classes and their relationships
  - Aid in the software development process



Reflection-Repository Trade-offs

	Reflection	Repository
(In)Completeness	Compilable	Reasonably parsable
(In)Efficiency	Lighter weight	Overkill for simple clients
(In)Consistency	Self representation	Could be out of phase
(In)flexibility	Can a tool designed to work with reflection run without a compiler?	Can a tool designed to work with files run using reflection?

Reflection-Repository Trade-offs

	Reflection	Repository
(In)Completeness	Compilable	Reasonably parsable
(In)Efficiency	Lighter weight	Overkill for simple clients
(In)Consistency	Self representation	Could be out of phase
(In)flexibility	Can a tool designed to work with reflection run without a compiler?	Can a tool designed to work with files run using reflection?

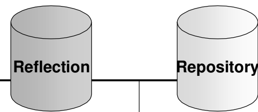
Reflection-Repository Trade-offs

	Reflection	Repository
(In)Completeness	Compilable	Reasonably parsable
(In)Efficiency	Lighter weight	Overkill for simple clients
(In)Consistency	Self representation	Could be out of phase
(In)flexibility	Can a tool designed to work with reflection run without a compiler?	Can a tool designed to work with files run using reflection?

Reflection-Repository Trade-offs

	Reflection	Repository
(In)Completeness	Compilable	Reasonably parsable
(In)Efficiency	Lighter weight	Overkill for simple clients
(In)Consistency	Self representation	Could be out of phase
(In)flexibility	Can a tool designed to work with reflection run without a compiler?	Can a tool designed to work with files run using reflection?

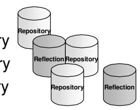
Reflection-Repository Trade-offs

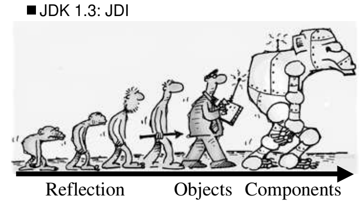


	Reflection	Repository
(In)Completeness	Compilable	Reasonably parsable
(In)Efficiency	Lighter weight	Overkill for simple clients
(In)Consistency	Self representation	Could be out of phase
(In)flexibility	Can a tool designed to work with reflection run without a compiler?	Can a tool designed to work with files run using reflection?

Reflection Evolutionary Path

- How to design client code that can work with current and future Reflective APIs with minimal change?
  - Scenario: a class browser tool
    - JDK 1.0: No reflection, must use repository
    - JDK 1.1: Use both reflection and repository
    - JDK 1.2: Doclet API can replace repository
    - JDK 1.3: JDI





Pluggability Approach

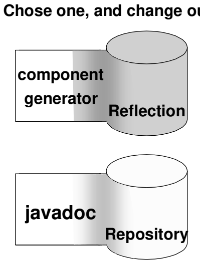
Pluggable Reflection

☛ **Pluggability: Ability to retarget clients to different meta-information sources**

- **There are many sources of meta-information**
  - **Different APIs**
    - Java.lang.reflect, com.sun.javadoc, com.sun.jdi
  - **Different versions of the same API**
    - JDK 1.0, JDK 1.1, JDK 1.2, JDK 1.3, JDK 1.4
  - **Different languages**
    - Java versus Smalltalk
  - **Different dialects of the same language**
    - Versions of Smalltalk
  - **Different reflective mechanisms**
    - Reflection versus AOP

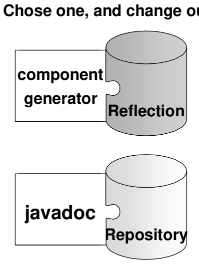
Pluggable Reflection

- **Ideally we could ignore the difference**
  - **Switch between sources freely**
  - **Chose one, and change our mind later**



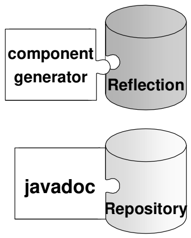
Pluggable Reflection

- **Ideally we could ignore the difference**
  - **Switch between sources freely**
  - **Chose one, and change our mind later**



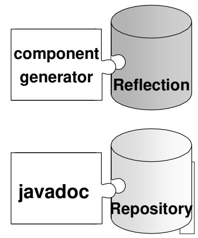
Pluggable Reflection

- Ideally we could ignore the difference
  - Switch between sources freely
  - Chose one, and change our mind later



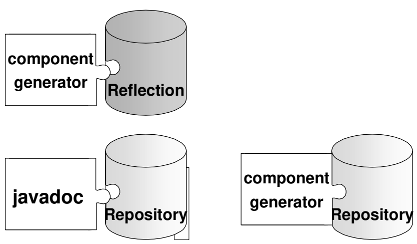
Pluggable Reflection

- Ideally we could ignore the difference
  - Switch between sources freely
  - Chose one, and change our mind later



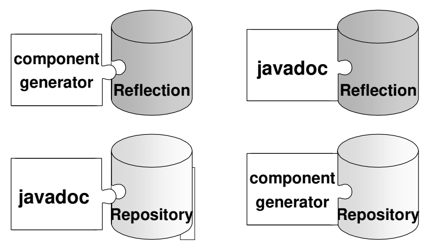
Pluggable Reflection

- Ideally we could ignore the difference
  - Switch between sources freely
  - Chose one, and change our mind later



Pluggable Reflection

- Ideally we could ignore the difference
  - Switch between sources freely
  - Chose one, and change our mind later



☞ Degree of Pluggability: Ease with which a tool can be retargeted

Extra-lingual Approaches

☞ Reconciliation via abstract representation

❶ Mirrored reflection using Interpreter pattern
❷ Visitor pattern approach
❸ Component approach

- Other and future approaches:
  - Lingual approaches
  - Retargeting using AOP

Pluggability through Mirror

Interpreter Approach

☛ **For retargeting legacy tools:**

❶ **Abstract grammar for reflection**

❷ **Generate a mirrored reflection**

- **Apply Interpreter pattern**

☛ **To retarget the tool:**

- **Change the import statements in the tool code to use the mirrored reflection, and recompile**
- **<u>Pros</u>:**
  - Useful for debugging reflective code
- **<u>Cons</u>:**
  - Identical interface

Grammar For Reflection

```
Class  ::=  name:String
            superclass:[Class]
            interfaces:Class[]
            Member[] ;

Member  ::=  Field | Routine | Class ;

Routine  ::=  Method | Constructor ;

Field  ::=  declaringClass:Class
            name:String
            type:Class ;

Method  ::=  declaringClass:Class
             name:String
             returnType:Class
             parameterTypes:Class[] ;

Constructor  ::=  declaringClass:Class
                  name:String
                  parameterTypes:Class[] ;
```


Meta-Information Client

A client of Reflection, or is it?

```
package program;
import java.util.Vector;
import java.lang.reflect.Field;

class MetaClient {
private void collectFields(Class cl,Vector result,Object host) {
   if (cl==null) return;
   Field[] declared = cl.getDeclaredFields();
   for(int i=0;i<declared.length;i++)
     try{
       result.add(declared[i].get(host));
     } catch (IllegalAccessException e) {}
   collectFields(cl.getSuperclass(),result,host);
 }
}
```

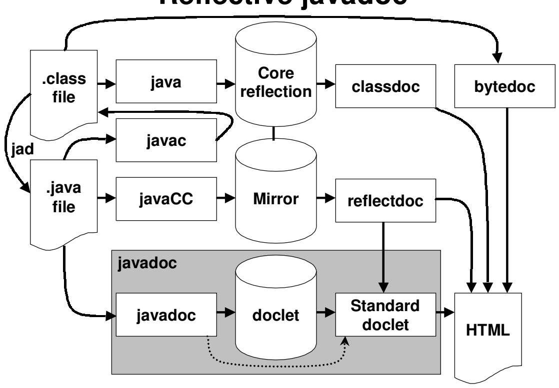

Meta-Information Client

A client of Reflection, or is it?

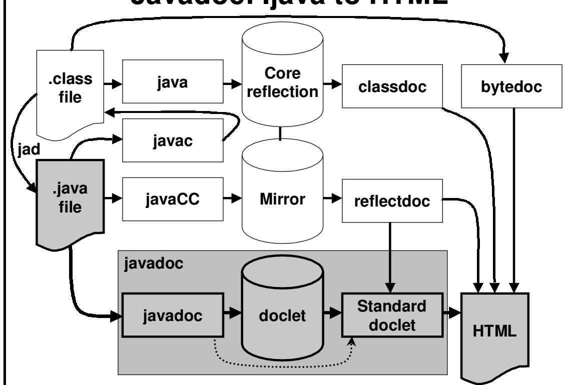
```
package program;
import java.util.Vector;
import edu.neu.ccs.mirror.java.lang.Class;
import edu.neu.ccs.mirror.java.lang.reflect.*;

class MetaClient {
private void collectFields(Class cl,Vector result,Object host) {
   if (cl==null) return;
   Field[] declared = cl.getDeclaredFields();
   for(int i=0;i<declared.length;i++)
     try{
       result.add(declared[i].get(host));
     } catch (IllegalAccessException e) {}
   collectFields(cl.getSuperclass(),result,host);
 }
}
```

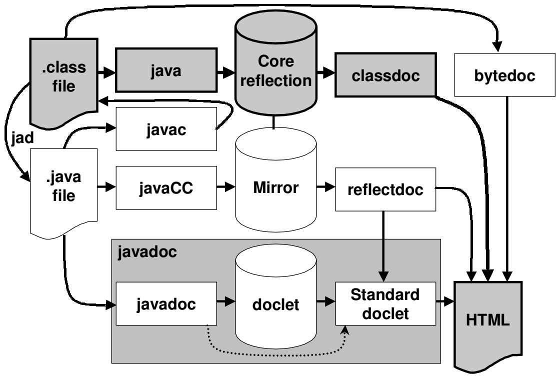

Reflective javadoc



Javadoc: .java to HTML

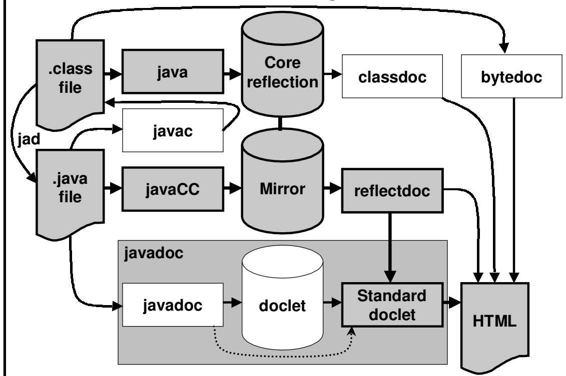


Classdoc: .class to HTML



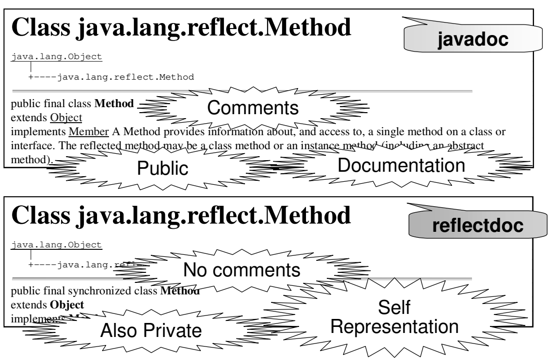
NU IBM

Reflectdoc: Either .java or .class



NU IBM

reflectdoc works with both



NU IBM

Pluggability through Components

Component Approach

☛ **For creating new tools that use reflection in a way that allow them to be retargeted:**

❶ **Generate ReflectionBeans**

- **Using a component generator** (*Lorenz & Vlissides. ICSE'01.)*
- **Input**: `java.lang.reflect`
- **Output**: `ReflectionBeans.jar`

❷ **Implement the tool using only the ReflectionBeans**

- **Instead of using reflection directly**

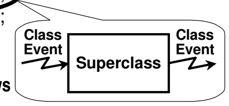
☛ **To retarget the tool:**

- **Change the `ReflectionBeans.jar` with an alternative implementation**
- **Pros**:
  - Implicit invocation
- **Cons**:
  - Event overhead

NU IBM

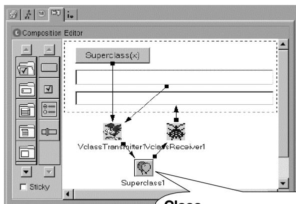
The Class "Class"

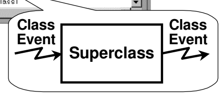
public final class Class **implements** Serializable{
 private Class() {}
 public static Class forName(String className)
  throws ClassNotFoundException {…}
  public Object newInstance() **throws**
   InstantiationException,IllegalAccessException{…}
  public native Class getSuperclass();
  public native Class[] getInterfaces();
  public Field[] getFields() **throws**
   SecurityException {…}
  public Method[] getMethods() **throws**
   SecurityException {…}
…
}



NU IBM

Example 1: Superclass

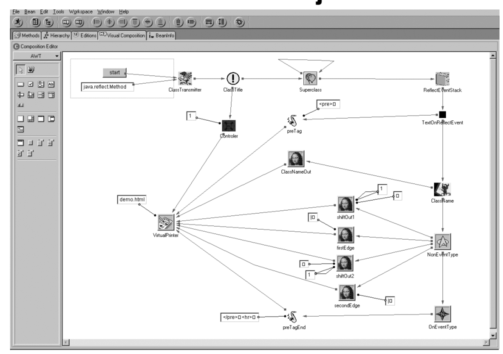




- **Superclass1**
  - **May use reflection or any other source of meta-information to find the super-class**

NU IBM

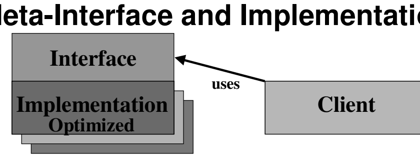
Example 2: JavaBeans Implementation of a Reflective javadoc



NU IBM

Conclusion, Related work, and Future work

Meta-Interface and Implementation



- **API: Interface independent of implementation**
  - **Evolution:** Client and implementation evolve separately
  - **Prototyping, Testing:** Choice of swappable implementations
  - **Portability:** Several back-ends

☞ **Pluggable Reflection decouples reflection and clients of meta-information.**

NU IBM

Main Points

- **Reflection is just an API that can have swappable implementations**
- **How to design client code that can work with alternative,current and future Reflective APIs with minimal change?**
- **Pluggability approaches**
  - **Legacy clients can be retargeted to use the mirrored hierarchy**
  - **New clients that use visitors as building block are pluggable at compile-time**
  - **For even looser coupling, third party clients may use components as their building blocks**

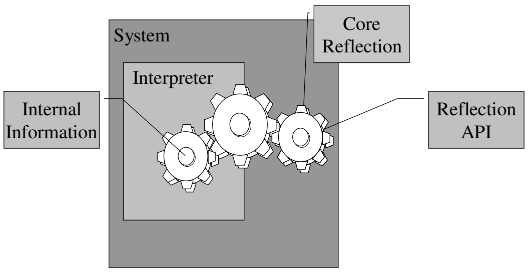
NU IBM

Related Work

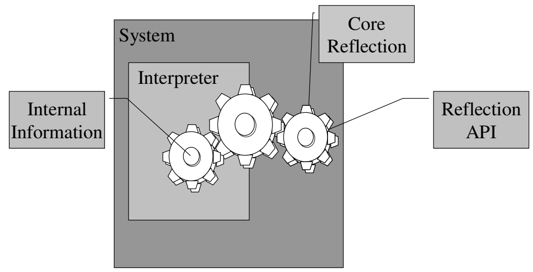
- **Self and Mirrors**
  - **Reflective computation (Smith)**
  - **Animorphic Smalltalk, Strongtalk (Bracha & Griswold)**
  - **Mirrors (Bracha & Ungar)**
- **Reflecting a language extension**
  - **ajdoc**
  - **archdoc**
- **Retargeting clients of reflection using AOP**
  - **AspectJ**
  - **HyperJ**
  - **Jiazzi**
- **AOP as an alternative source of meta-information**
  - **Refection as an aspect**

NU IBM

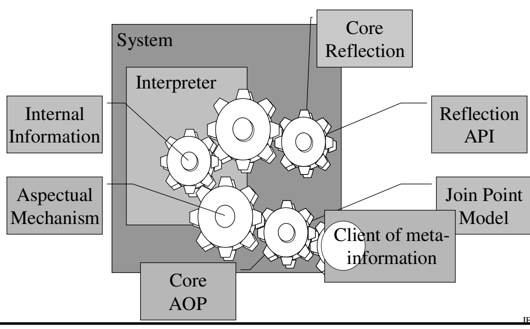
Future Work



Future Work



Future Work



The End

ICSE 2003

More Information:

- **URL:** http://www.ccs.neu.edu/home/lorenz/papers/icse03/
- **Please send comments to:** lorenz@ccs.neu.edu
  vlis@us.ibm.com