An Introduction to Reflective Java

Zhixue Wu

APM Ltd.

18 Oct. 1998

Motivation

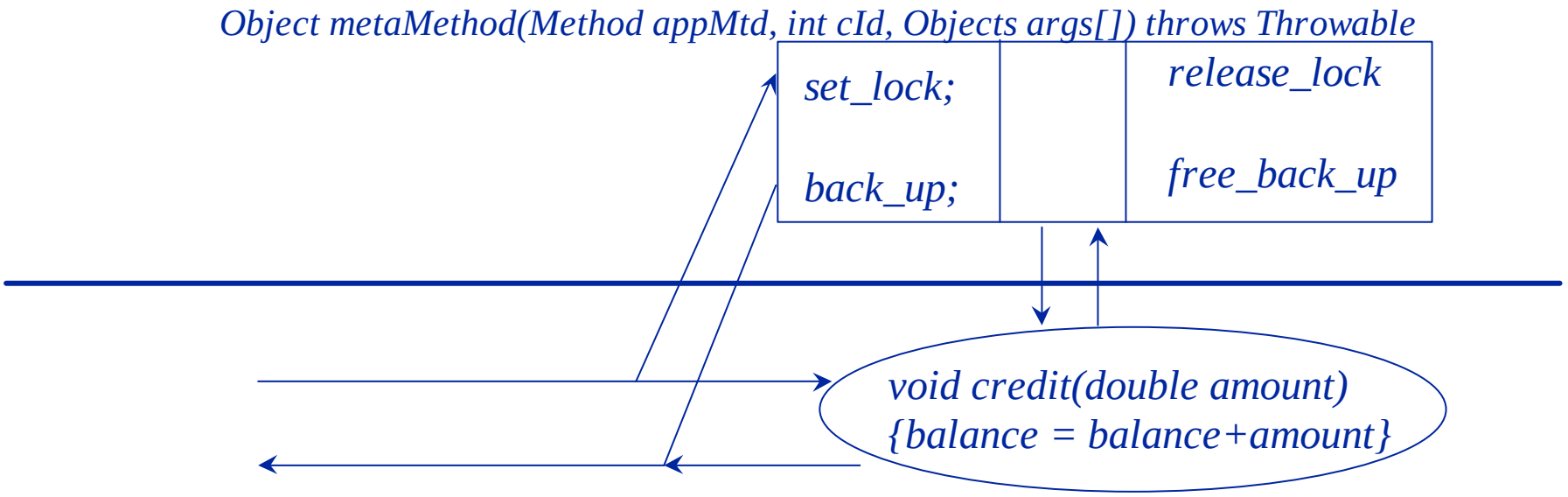
- Java Advantages
  - object oriented
    - separate interface from implementation
  - Portable
    - write once, run anywhere
  - Dynamic
    - dynamic loading and linking
- Issues
  - Make a program portable to every infrastructure
  - Enable a system to adjust its behaviour according to environment
  - Allow application deployers to configure system capabilities

Reflective Java

- Enable Java-powered system to be customised according to particular requirements of applications and run-time environment
  - statically at compile time and dynamically at run-time
  - flexibly
  - transparently
- Make Java reflective
  - without any change to the language itself
  - without any change to its compiler
  - without any change to its virtual machine

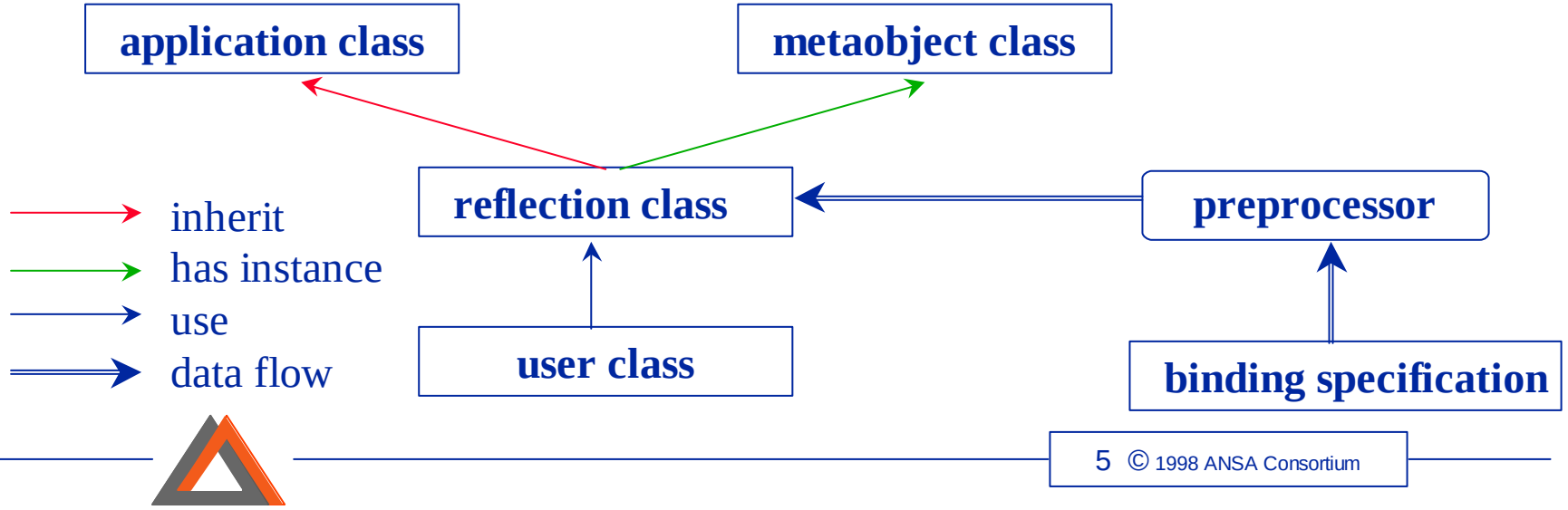
Reflective Method Invocation

- Method invocation is intercepted by a metaobject
- Extra processing can be done before and after method execution
- Meta information for classes, objects, and parameters is accessible
- Values of parameters can be manipulated at meta level



Building Process

- Application classes are implemented by application developers
- Metaobject classes are implemented by system developers
- End-users specify metaobject binding via a simple script language
- A preprocessor is used to generates reflection classes
- End-user application performs functions through the reflection class



Binding Specification

- Binding specification describes the association between an application class and a metaobject class
- When being created, an application object will be bound to an metaobject automatically
- The binding can be changed dynamically at run-time

```
import transaction.*;

refl_class Account: Meta_Lock{
 public Account(String nm):WRITELOCK;
 public void init(String nm, double mm):WRITELOCK;
 public void credit(double mm):WRITELOCK;
 public void debit(double mm) throws OverdrawException:WRITELOCK;
 public double balance():READLOCK;
}
```

Metaobject Implementation

```
public ConcurrencyMetaobject extend Metaobject {
  public ConcurrencyMetaobject(Object appObject)
  {
    super(appObject);
    lock = new Lock();
  }

  public Object metaMethod(Method appMtd, int cId, Object[] args)
       throws Throwable
  {
    if(cId==READLOCK) lock.set_read_lock(); else set_write_lock();

    Object rslt = appMtd.invoke(appObject, mArgs);

    if(cId==READLOCK) lock.release_read_lock(); else release_write_lock();

    return rslt;
  }

  protected Lock lock;
}
```

Dynamic Binding

- Java allows loading class and constructing objects dynamically
- Reflective Java provides a pair of operations to check and set the metaobject of an object
- Application program thus can change metaobject binding dynamically via these operations

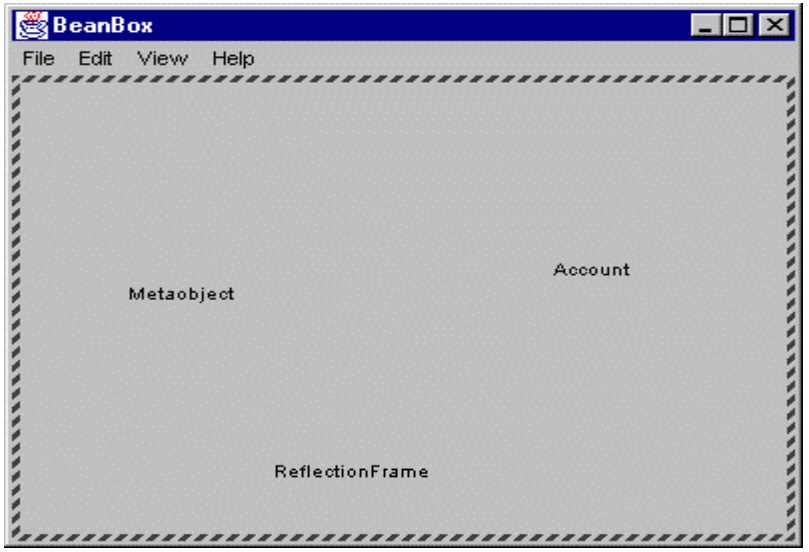
```
public Metaobject getMetaobject( ){
       return metaobject;
}

public void setMetaobject(String clsName) throws ClassNotFoundException
{
   metaobject = (Metaobject)Class.forName(clsName).newInstance();
   metaobject.init(this);
}
```

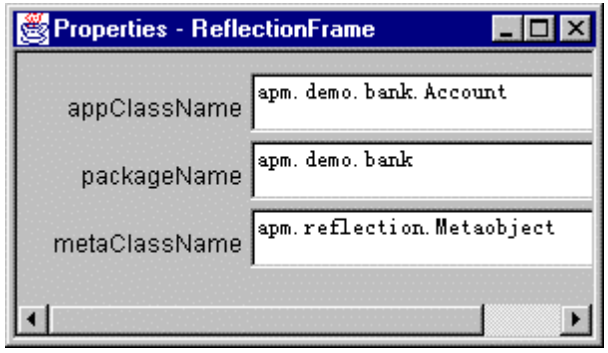
Practical Issues

- Better development environment
  - dislike a separate binding specification
  - dislike an extra preprocessing step
- Hard to provide multiple capabilities
  - multiple metaobject binding
  - metaobject reuse
- Consistency concerns in dynamic binding
  - between old and new metaobjects
  - system states
- Difficulty of implementing metaobjects
  - generic programming
- Performance
  - normal story

Building Toolbox

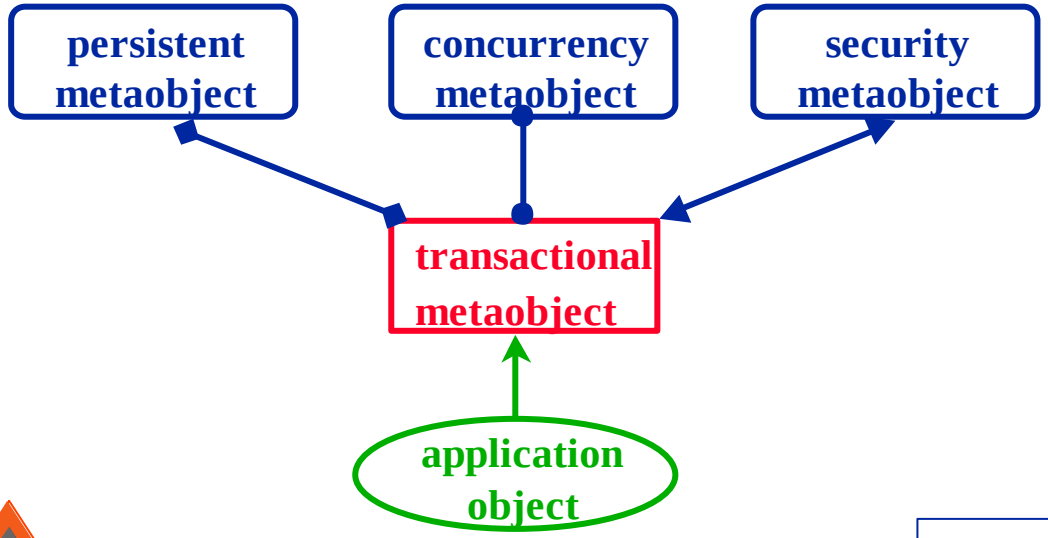
- A graphical developer environment
- Interactive input: selecting instead of retyping
- Integrated preprocessor and compiler: byte code output



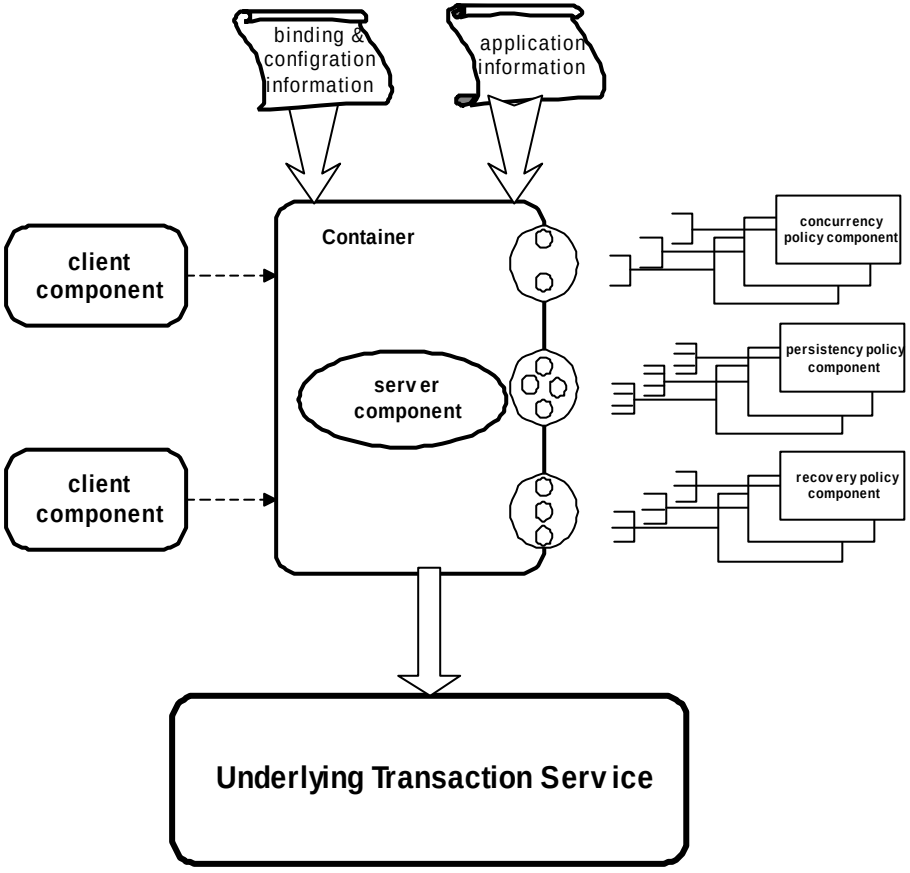


Two-Layer Approach for Metaobjects

- First-layer metaobject:
  - interception method invocation
  - coordinator second-layer metaobjects
- Second-layer metaobjects:
  - provide a particular capability
- Contract interface between first and second layer metaobjects
  - ensure a second layer metaobject reusable
  - ensure consistence between new and old metaobject



A Reflective Transaction Architecture



- Container provides a first layer metaobject: *transactionalMetaobejct*
- Three second-layer metaobject interfaces: *persistency*, *concurrency*, *recovery*
- Each interface may have multiple implementations
- Application deployer choose metaobjects for a application
- "Off-the-shelf" metaobjects can be used

Summary

- Pure Java
- Simple, but powerful
- Dynamic binding
- User friendly development environment
- Ensure compatibility between metaobjects
- Two-layer approach on constructing metaobjects
- Reasonable performance

- *Issue: how to ensure system state consistency*