Introduction to Reflective Java

Zhixue Wu & Scarlet Schwiderski

APM Ltd.

12 Nov. 1997

Observations

- Requirements
  - The one size fits all design strategy becomes obsolete
    - mobile computing, internet programming, and mutimedia applications
    - different considerations and requirements
  - System software must be flexible and customisable at runtime
    - many attributes of the application environment vary from time to time, and from place to place
- Technologies
  - Object-oriented programming has suggested methods for building flexible system software components
    - Java, Java Beans
    - reflection and metaobject protocol (MOP)
  - It’s time to transfer these ideas to mature technology

Java Advantages

- A simple, object oriented, distributed, interpreted, robust, safe, architecture neutral, portable, high performance, multithreaded, dynamic language
- Important ones
  - object oriented
    - separate interface from implementation
  - Portable
    - write once, run anywhere
  - Dynamic
    - dynamic loading and linking

Functional and Non-functional Capabilities

- Functional capabilities are primarily concerned with the purpose of an application, that is, the business logic
- Non-functional capabilities are more concerned with the fitness of the application to the run time environment, that is, the system issues

Java Problems: Not Flexible Enough

- Approach
  - Java takes the API approach to provide non-functional capabilities
  - API only implements a fixed, single point in the whole design space
  - Application cannot be decoupled from the choice of non-functional capabilities
  - Changing non-functional capabilities requires changing the source code of the application
- Result
  - A program is not portable to every infrastructure
  - A system cannot adjust its behaviour according to conditions
  - The execution side cannot control the choice of non-functional capabilities of an applet

Reflective Java

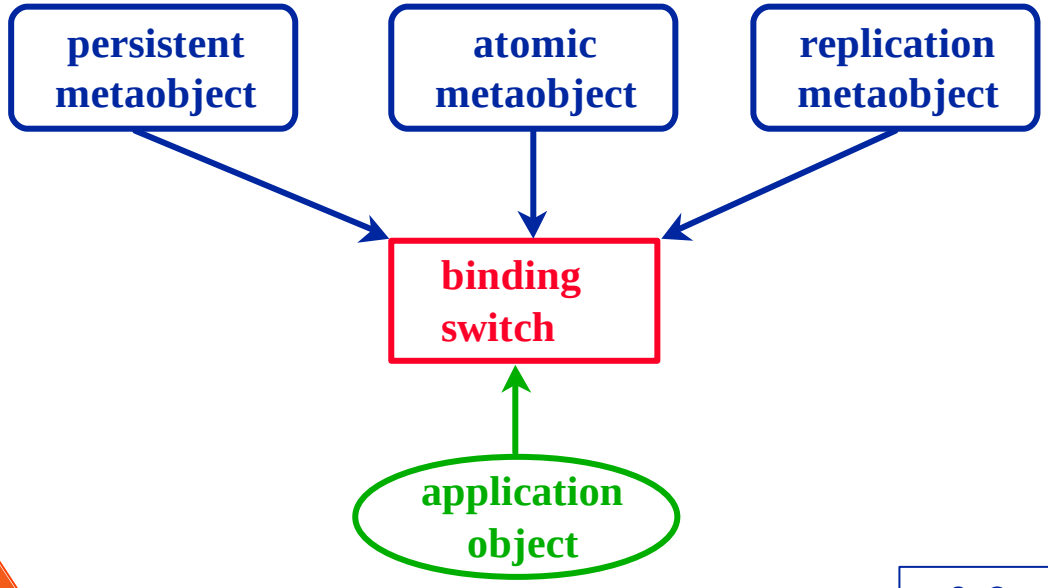
- Enable Java-powered system to be customised according to particular requirements of applications and run-time environment
  - statically at compile time and dynamically at run-time
  - flexibly
  - transparently
- Make Java reflective
  - without any change to the language itself
  - without any change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

Reflection and Metaobject Protocol

- Reflection
  - the capability of a system to reason about and act upon itself and adjust itself to changing conditions
  - opens up a system's implementation in a principled way
  - provides an abstraction of the system's behaviour and internal state at the meta level
- Metaobject protocol = reflection + object-oriented programming
  - represents the system at the meta level in a family of meta objects
  - allows the system's behaviour to be locally and incrementally adjusted

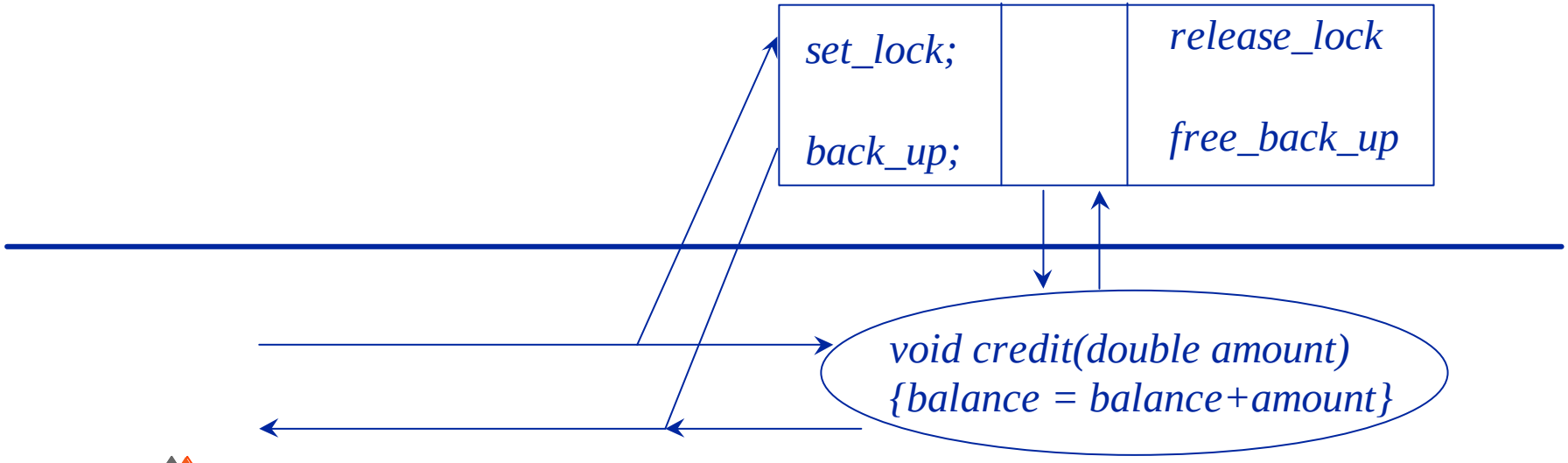
Core Idea

- Functional requirements are satisfied by application objects
- Non-functional requirements are satisfied by metaobjects
- Non-functional capabilities are added to an application object by binding it to an appropriate metaobject
- actual behaviour of an object can be changed by meta binding



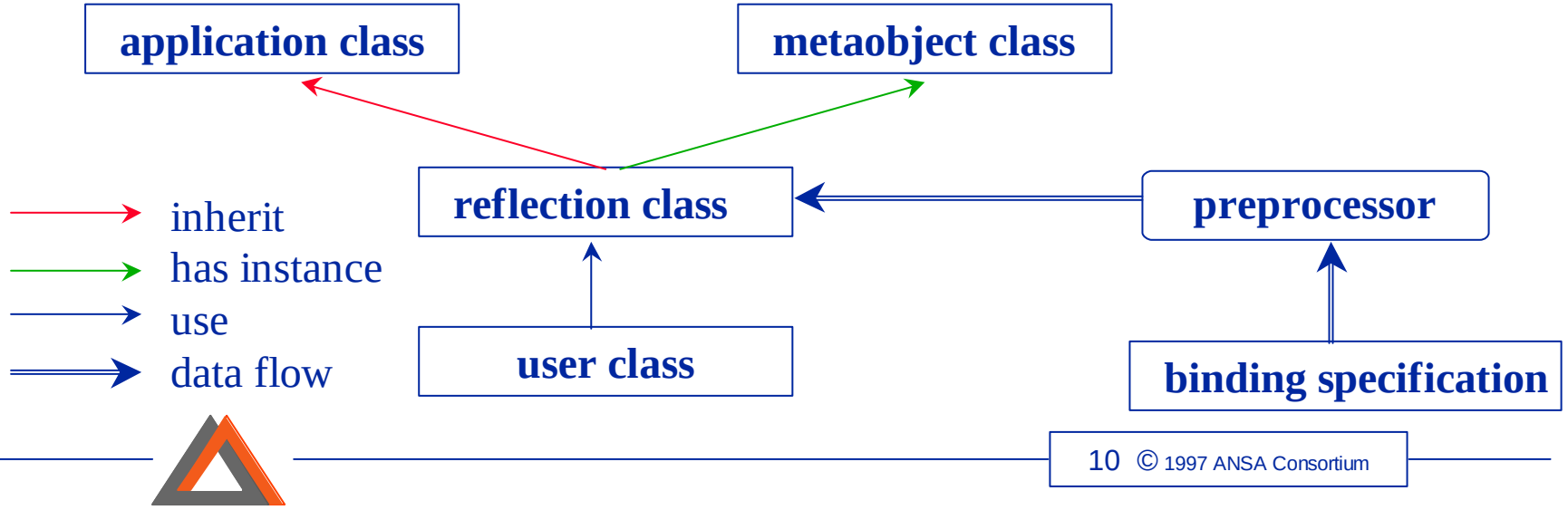
Reflective Method Invocation

- Method invocations are interceptable and changeable by users
  - metaBefore operation
  - metaAfter operation
- Meta data for classes, objects, and parameters is accessible at meta level
- 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 describe which non-functional capability should be added to an application object through a simple script language
- A preprocessor generates a reflection class
- 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):201;
 public void init(String nm, double mm):201;
 public void credie(double mm):201
 public void debit(double mm) throws OverdrawException:201;
 public double balance():202;
}
```

Metaobject Implementation

An Application Class

```
class Account{

 public void credit(double mm)
  {balance = balance + mm;}

 public void debit(double mm)
      throws Overdraw{
   if(balance < mm)
              throw new Overdraw();
   balance = balance - mm;
}

public double check()
  { return balance;}

private double balance;
}
```

A Meta Class

```
class Meta_Lock extends Metaobject{
 public void MetaBefore(MID mid,
        CID cid, Arg arg) {
   if(cid==201) //read operation
              set_read_lock();
   else set_write_lock();
 }

 public void MetaAfter(MID mid,
        CID cid, Arg arg) {
   if(cid==201)
              release_read_lock();
   else relese_write_lock();
 }

}
```

Benefits

- Easy to upgrade a product to adapt to changes and new services, either in hardware or application requirements
- Flexibility to customise policies dynamically to suit the run-time environment
- High level transparency to applications
- Write an application once, run it any time, anywhere, in any environment
- Free choice of components and flexible configurations
- Better services for your customers, provided faster and at lower cost

Deliverable

- Preprocessor
  - generating reflection classes from binding specifications
- Metaobject package
  - Metaobject class
  - classes used by Metaobject class
- Demos
  - simple bank demo
  - data display demo
  - other test demos
- A prototyping persistent system
  - a persistent service implementation using Reflective Java
- A prototyping transaction system
  - a transaction service implementation using Reflective Java