APM

POSEIDON HOUSE • CASTLE PARK • CAMBRIDGE • CB3 0RD UNITED KINGDOM
+44 1223 515010 • Fax: +44 1223 359779 • Email: apm@ansa.co.uk • URL: http://www.ansa.co.uk

ANSA Phase III

Refletive Java: The Design, Implementation and Applications

Zhixue Wu and Scarlet Schwiderski

Abstract

TThis is a presentation document for Reflective Java.

The purpose of Java++ is to make some features of Java reflective, thus enabling Java-powered system to be customised dynamically, flexibly and transparently to suit a particular application.

Method calls are made open-ended; a simple pre-processor that translates Java++ programs into standard Java program and generates classes for binding an object to its metaobject.

APM.1877.02 **Approved** 5th December 1996
External Paper

Distribution:
Supersedes:
Superseded by:

Reflective Java

The Design, Implementation and Applications

Zhixue Wu & Scarlet Schwiderski

Observations

Requirements:

- The one size fits all design strategy becomes obsolete
  - mobile computing, internet programming, and multimedia applications
  - different considerations and requirements
- System software must be made flexible and customisable at run-time
  - many attributes of the application environment vary from time to time, and from place to place

Technologies:

- Object-oriented programming and language theory has suggested methods for building flexible system software components
  - Java
  - 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:

- Object Oriented
  - separate interface from implementation
- Portable
  - write once, run anywhere
- Dynamic
  - dynamic loading and linking

Java Problems: Not Flexible Enough

- 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

- A program is not portable to every infrastructure
- A system cannot adjust its behaviour according to conditions
- The execution site cannot control the choice of non-functional capabilities of applets

Functional and Non-Functional Capabilities

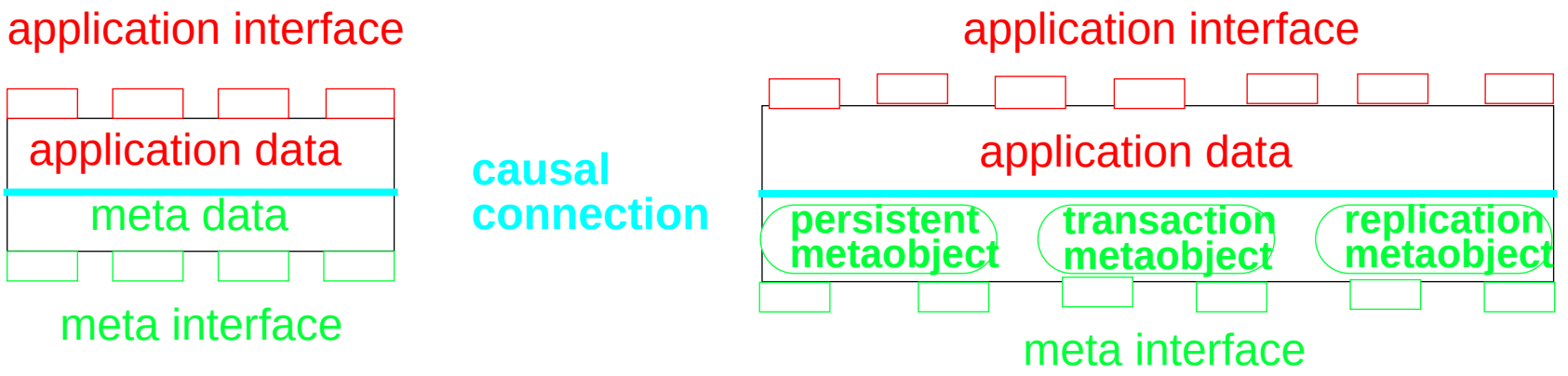
- Functional capabilities are primarily concerned with the purpose of an application, i.e. what it does
- Non-functional capabilities are more concerned with its fitness to fulfil its purpose, i.e. how well it does it
  - distribution
  - fault tolerance
  - persistence
  - concurrency control

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

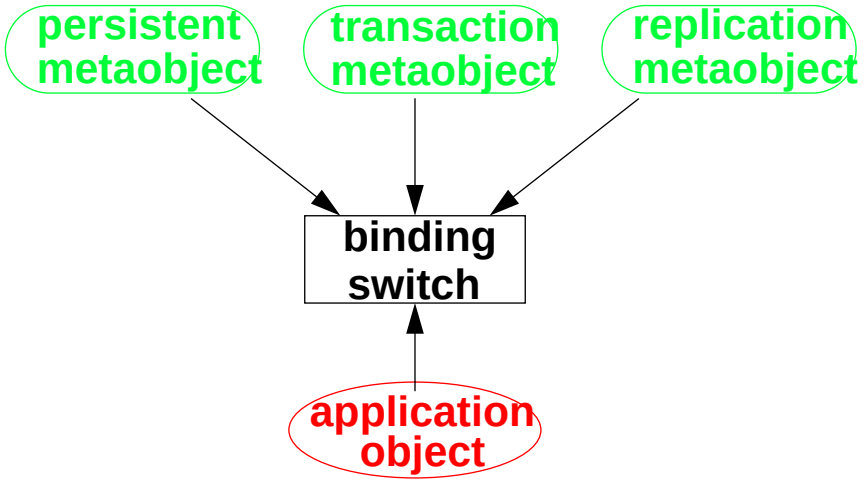
Reflection and Metaobject Protocol

- **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 using a family of meta objects
  - allows the system's behaviour to be locally and incrementally adjusted



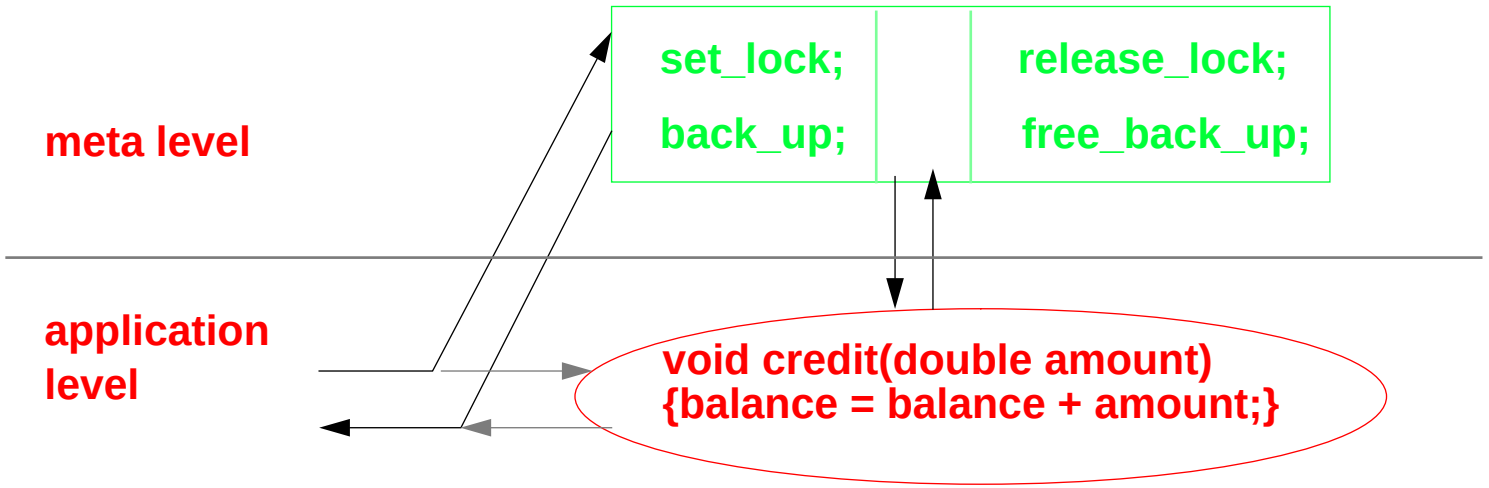
Idea

- 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 application object can be changed by binding it to a different metaobject



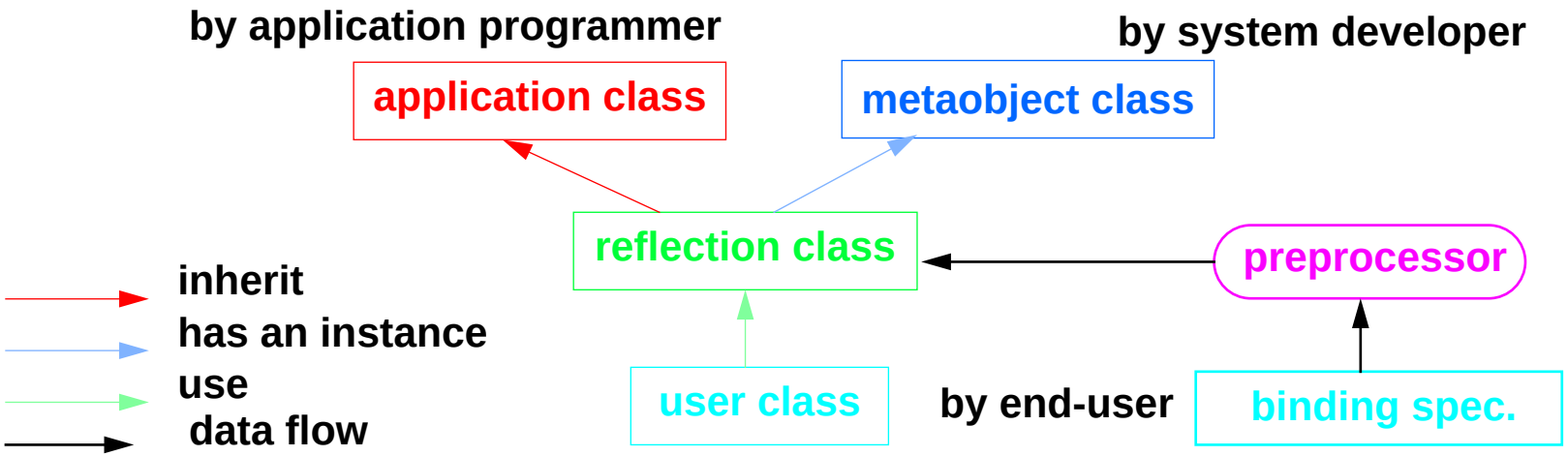
Reflective Method Invocation

- 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



Implementation

- 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 through a simple script language
- A preprocessor generates a reflection class
- The end-user application performs functions through the reflection class



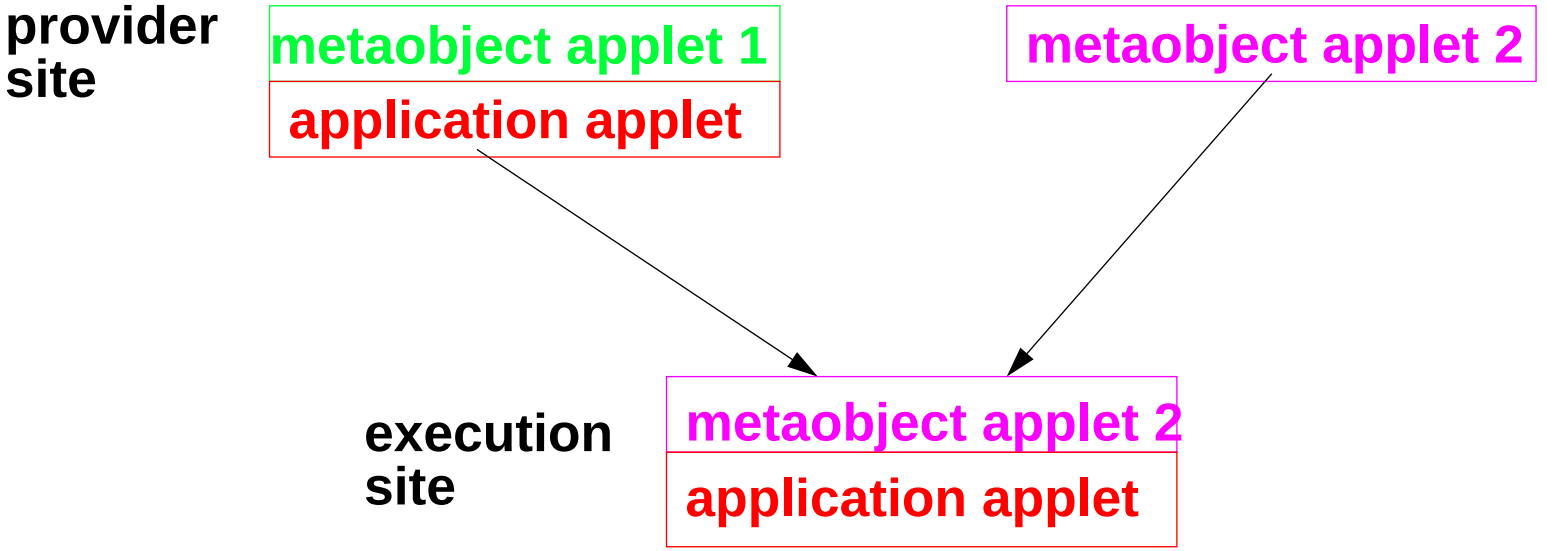
Binding Specification

- **Binding specification describes the association between an application class and a metaobject class**
- **When being created, an instance of an application class will be bound to an instance of the corresponding meta class automatically**
- **The binding can be changed dynamically at run-time without stopping the program**

```
import transaction.*;

refl_class Account : Meta_Lock {
   public Account(String nm) throws Throwable:1;
   public void init(String nm, double amt):201;
   public void credit(Control ctl, double mm):201;
   public void debit(Control ctl, double mm)  throws OverdrawException:201;
   public double check(Control ctl) :202;
}
```

Context Sensitive Applets

- Downloaded application applet is attached to a metaobject selected by the applet provider
- The attached metaobject can be replaced with another metaobject at the execution site in order to adapt to the particular environment



Simple Bank Demo

Application class

```
class Account {

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


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

 public double check( )
  { return balance;}


 private double balance;
 }
```

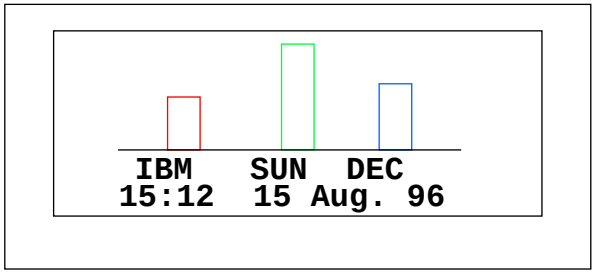
Metaobject class

```
class Meta_Lock
          extends MetaObject {

public void metaBefore(MID mid,
              CID cid, Arg arg)
    { if (cid == 201)
        set_read_lock();
      else set_write_lock();
    }

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


  synchronized void set_read_lock();
  synchronized void set_write_lock();
}
```

Simple Stock Demo

Simple Stock Applet

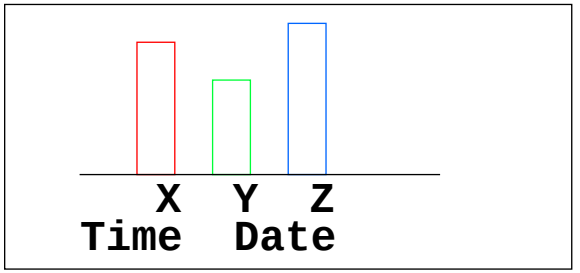
```
IBM     123.98
SUN     234.65
DEC     167.73

15:12   15 Aug. 96
```



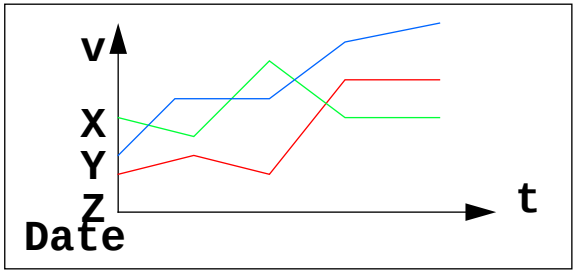

```
IBM     123.98
SUN     234.65
DEC     167.73

15:12   15 Aug. 96
```

Metaobject Classes

Display Applet A



Display Applet B



Advantage

Write a Java application once,

run it anytime, anywhere,

in any environment,

with any “-ability”

Benefits

- Easy to upgrade product in order to adapt to changes: either in hardware or in application requirements
- Flexibility to customise policies dynamically to suit run-time environment
- High-level transparency to applications
- Free choice of components
- Flexible configuration