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

Java++ presentation

Zhixue Wu and Scarlet Schwiderski

Abstract

TThis is a presentation document for Java++.

The purpose of Java++ is to make Java reflective so that it becomes easier for a Java-powered system to provide non-functional capabilities to its applications transparently and flexibly. At the same time, it makes it easier for application software to adjust its behaviour to meet new requirements.

The project will provide a reflective language Java++, a variant of the Java language, in which 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.1822.00.01 **Draft** 14th August 1996

External Paper

Distribution:

Supersedes:

Superseded by:

Java++:

Make Java More Adaptable

Zhixue Wu & Scarlet Schwiderski

The Challenge

- Configure the non-functional capabilities of applications without changing the source code
- 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
  - the execution site cannot control the choice of non-functional capabilities of applets
- Java enables a program to be portable to any platform, but not to any software environment

The Risk

- Complex “one size fits all” APIs with poor performance emerge
- Application programmers have to rewrite their source code in order to:
  - meet new application demands
  - port it to new environments
- Applets cannot be adapted to the environment of the execution site
  - limits utility of downloaded code
- The portability provided by Java cannot be fully exploited

The Key Issue

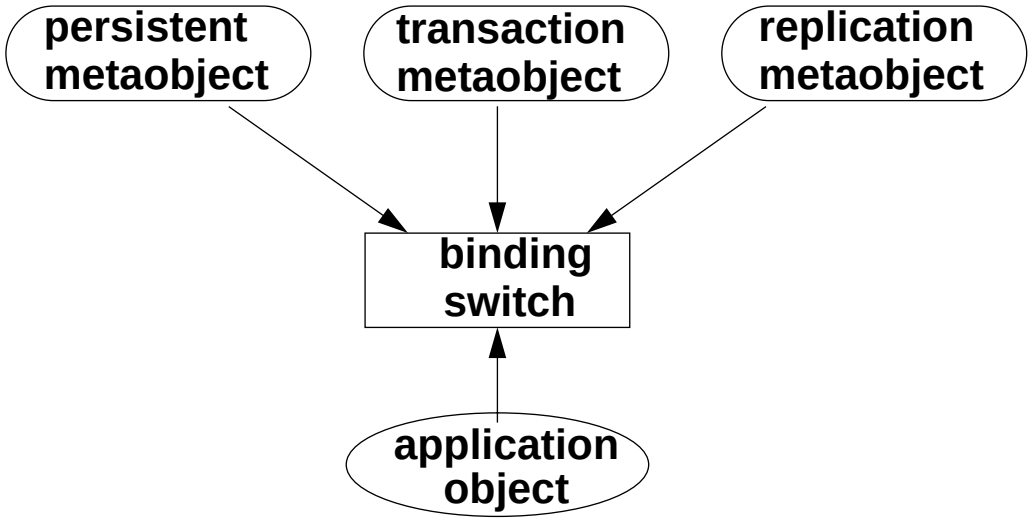
- **Finding a flexible approach to provide non-functional capabilities to Java-powered applications**

Options

- Making changes to the Java language
- Using object-oriented technologies, such as inheritance
- Making Java reflective
  - without any change to the language itself
  - without any change to its compiler
  - taking the MetaObject Protocol (MOP) approach

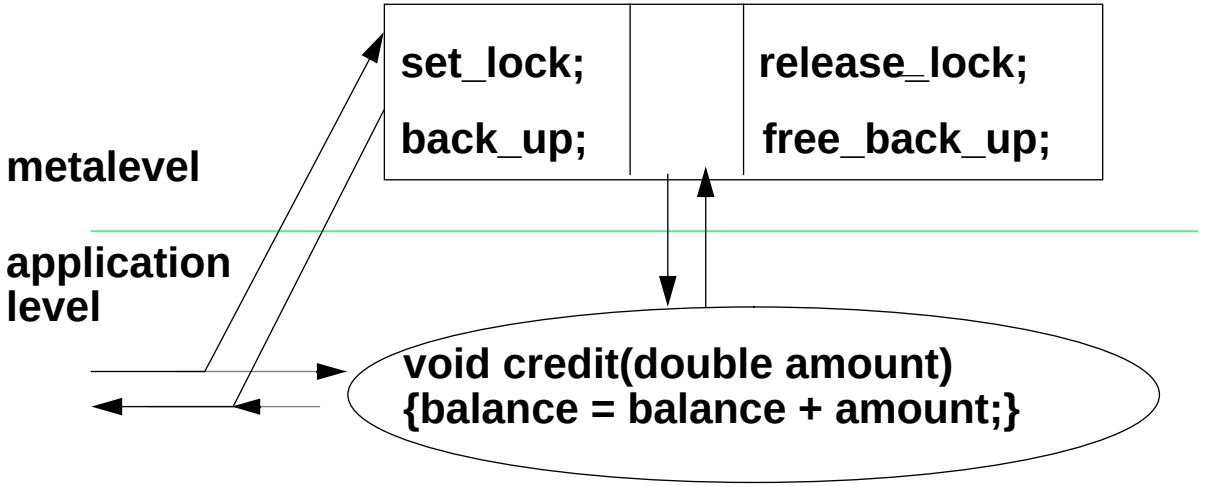
The MOP Approach

- Applications are implemented as normal objects
- Non-functional capabilities are implemented as 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



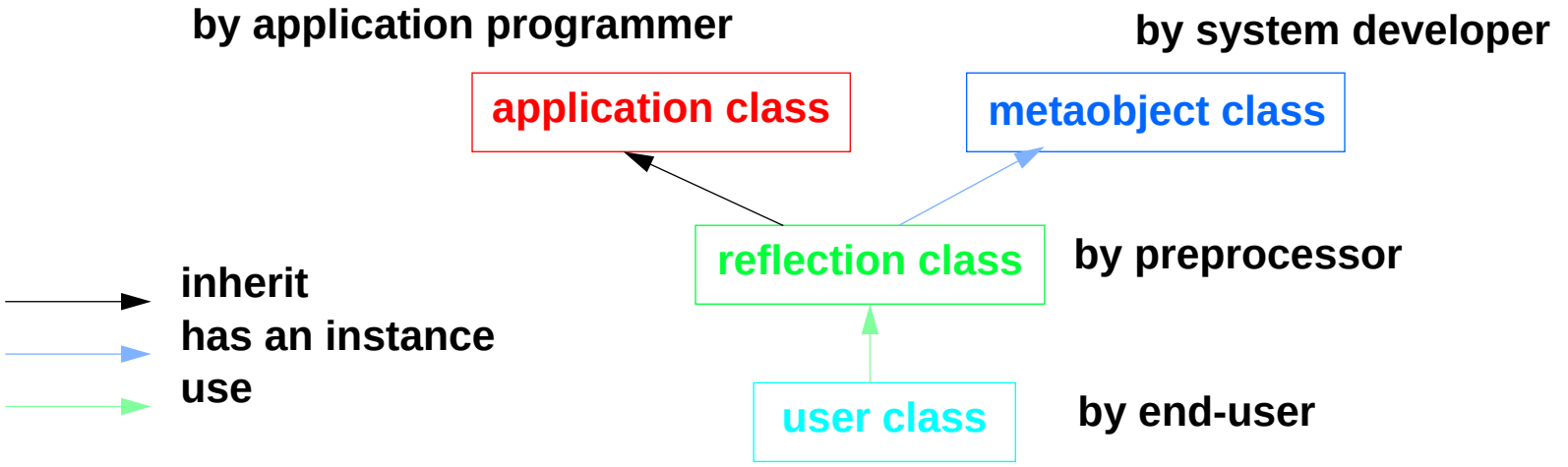
Making Java Reflective

- **Method calls are interceptable**
- **Users can redefine the behaviour of method calls through metaobjects**



Building Reflective Systems

- 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 class
- A preprocessor generates a reflection class
- The end-user application performs functions through the reflection class transparently



Example

Application class

```
class Account {
public:
 void credit(double m)
 { balance = balance + m};

 double check( )
 { return balance};

private:
 double balance;
}
```

Binding specification

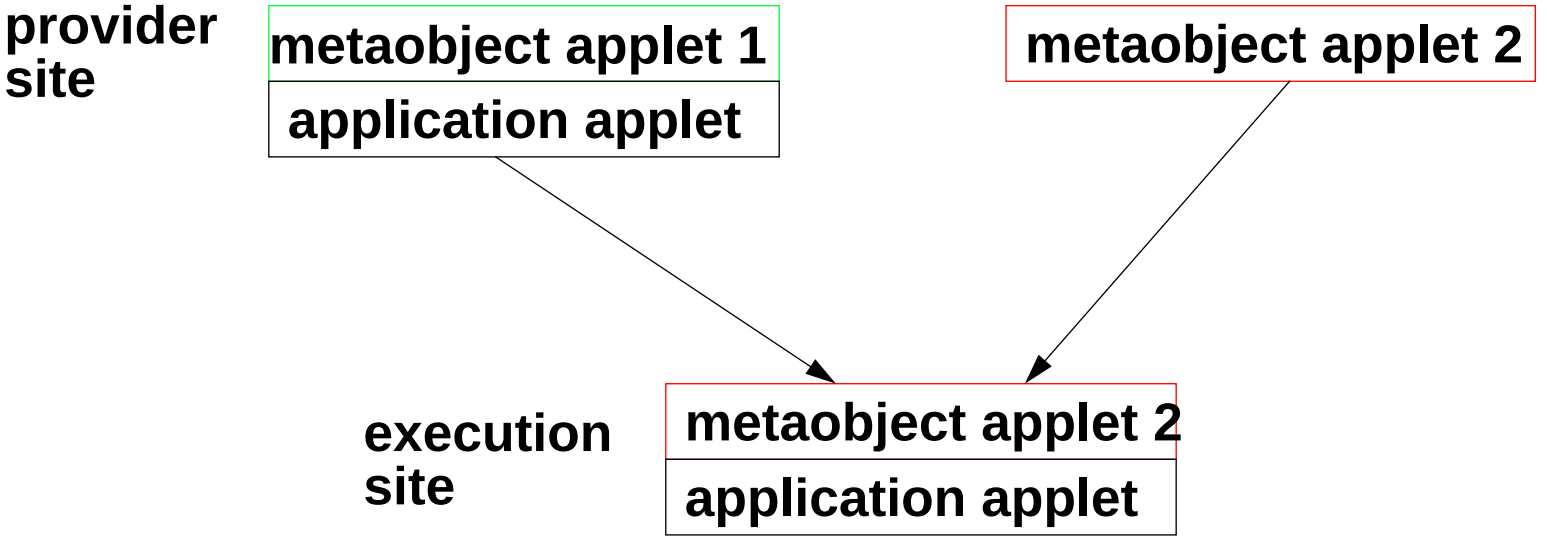
```
class Account : Meta_lock
{
 void credit(double):0;
 void check( ):1;
}
```

Metaobject class

```
class Meta_class Meta_Lock
          extends MetaObject {
public:
  void Meta_Before(MID mid,
              CID cid, Arg arg)
    { if (cid == 1)
        set_read_lock();
      else set_write_lock();
    };

  void Meta_After(MID mid,
                CID cid, Arg arg)
      { if (cid == 1)
          release_read_lock();
        else release_write_lock();
      };

private:
   synchronized set_read_lock();
   synchronized set_write_lock();
}
```

Context Sensitive Applets

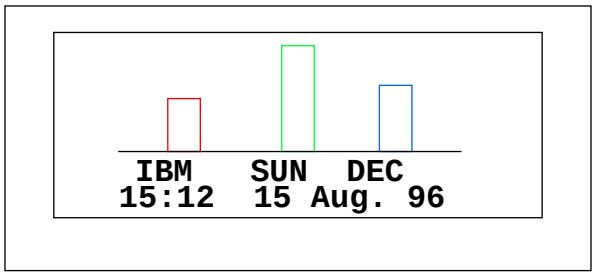
- Downloaded application applet is attached to a metaobject selected by the applet provider
- The attached metaobject can be replaced with another metaobject at the execution site in order to adapt to the particular environment



Applet Example

Simple Stock Applet

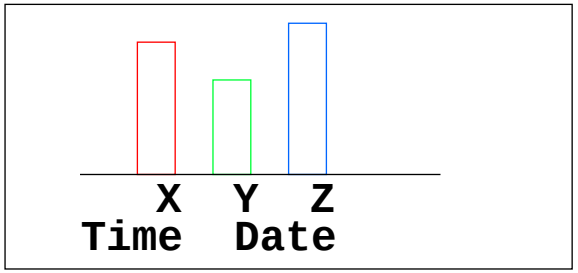
```
IBM     123.98
SUN     234.65
DEC     167.73

15:12   15 Aug. 96
```



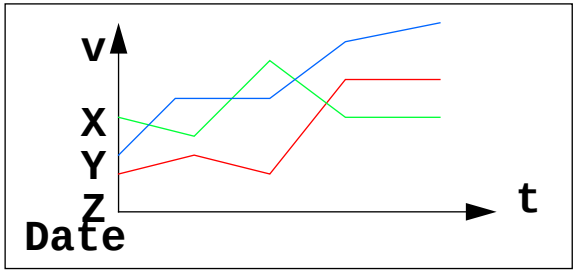

```
IBM     123.98
SUN     234.65
DEC     167.73

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 non-functional capability

Benefits

- **Benefits to system developers**
  - freedom to change implementation strategies for non-functional capabilities
  - easy to maintain product in order to meet new demands
  - high-level transparency
- **Benefits to application developers**
  - concentrate on functional requirements
  - easy to make changes to implementation
- **Benefits to end-users**
  - applet can be customised dynamically to suit environment
  - freedom to choose how to provide own non-functional capabilities
  - easy to add non-functional capabilities to applications