Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (01223) 515010**
INTERNATIONAL: **+44 1223 515010**
FAX: **+44 1223 359779**
E-MAIL: **apm@ansa.co.uk**

ANSA Phase III

Reflective Java and Its Applications

Zhixue Wu

Abstract

This is a presentation of Reflective Java for the DeVa workshop.

The presentation describes the general features of Reflective Java at first, and then focuses on showing how Reflective Java can be used to solve some important issues raised by the DeVa project.

APM.1971.01 **Approved** 17th March 1997

External Paper

Distribution:
Supersedes:
Superseded by:

Reflective Java:
A Useful Tool for Design for Validation

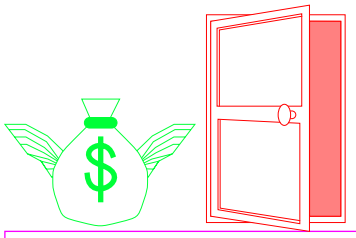
Zhixue Wu

APM Ltd.

20th March 1997

Context

- **A reflective language is a basic requirement for DeVa**



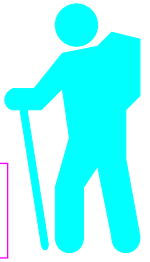
DeVa

Conceptual Framework

Separation of Concerns

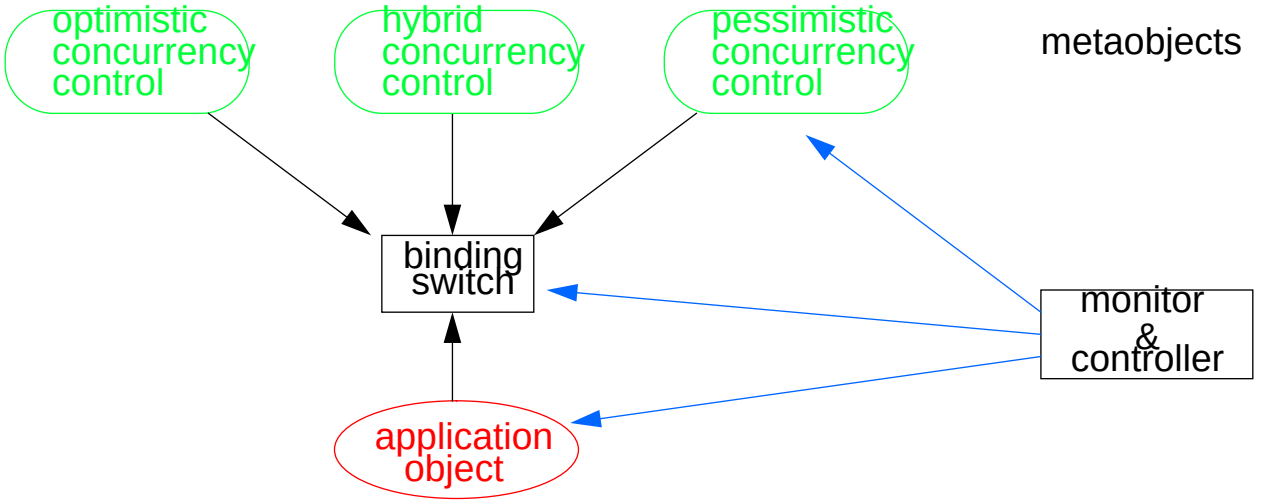
Reflection and Metaobject Protocols

Object-Oriented Reflective Programming Language



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
- Customisation can be done by switching a binding between an application object and its metaobject
- A clear separation can be made between functional and non-functional components



Requirements for a Reflective Language

- Separating binding specification from implementation
- Providing both static and dynamic bindings
- Allowing auxiliary information to pass to metalevel
- Supporting object-oriented programming
- Based on an easy and popular language

Current Situation

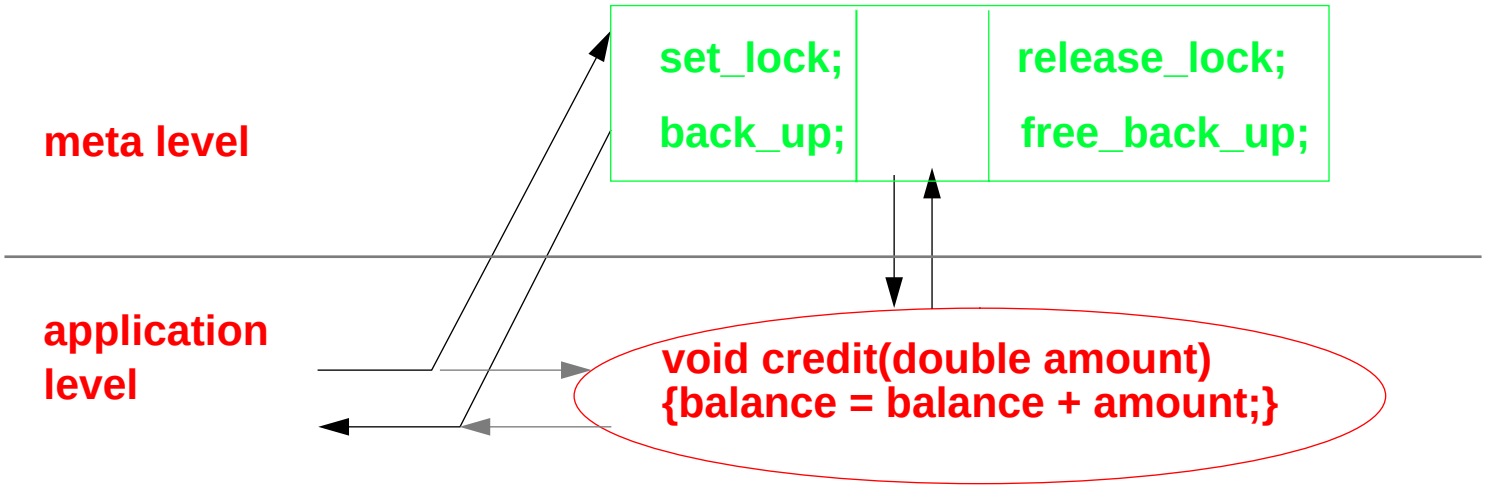
- **Interpreted languages: SmallTalk, CLOS, BETA, ABCL**
  - **supporting dynamic binding**
  - **not popular for system developing**
- **Compiled languages: Open C++**
  - **based on a popular language**
  - **no dynamic binding**
  - **binding specifications are mixed in the implementation code**
- **No one supports for providing auxiliary information**

Reflective Java

- Java: a simple, object oriented, distributed, interpreted, robust, safe, architecture neutral, portable, high performance, multithreaded, dynamic language
- Advantages
  - object-oriented: separate interface from implementation
  - architecture neutral: write once, run anywhere
  - dynamic loading and linking
  - simple and familiar
- Reflective Java: make Java reflective
  - without any change to the language itself
  - without any change to its compiler
  - without any change to its virtual machine

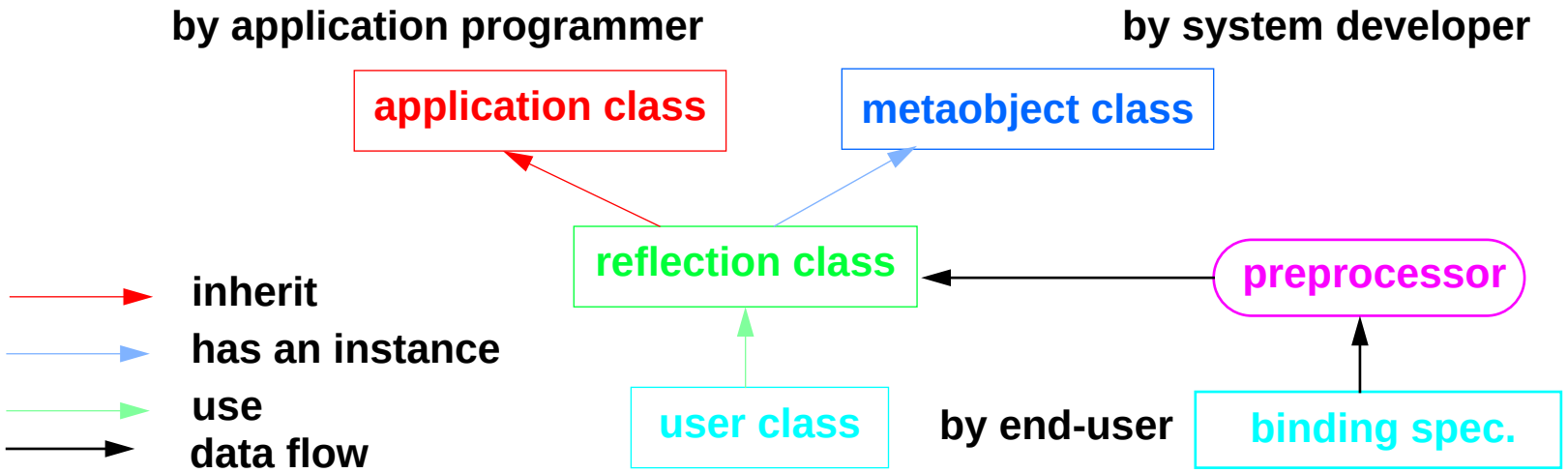
Reflective Method Invocation

- Method invocations are interceptable and changeable by users
  - metaBefore operation
  - metaAfter operation
- Meta data for classes, objects 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 declarative language
- A preprocessor generates a reflection class
- The end-user application performs functions through the reflection class



Binding Specification

- **Describes an 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**
- **The specification is separated from the implementation of objects**

```
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;
}
```

Simple Example

Application class

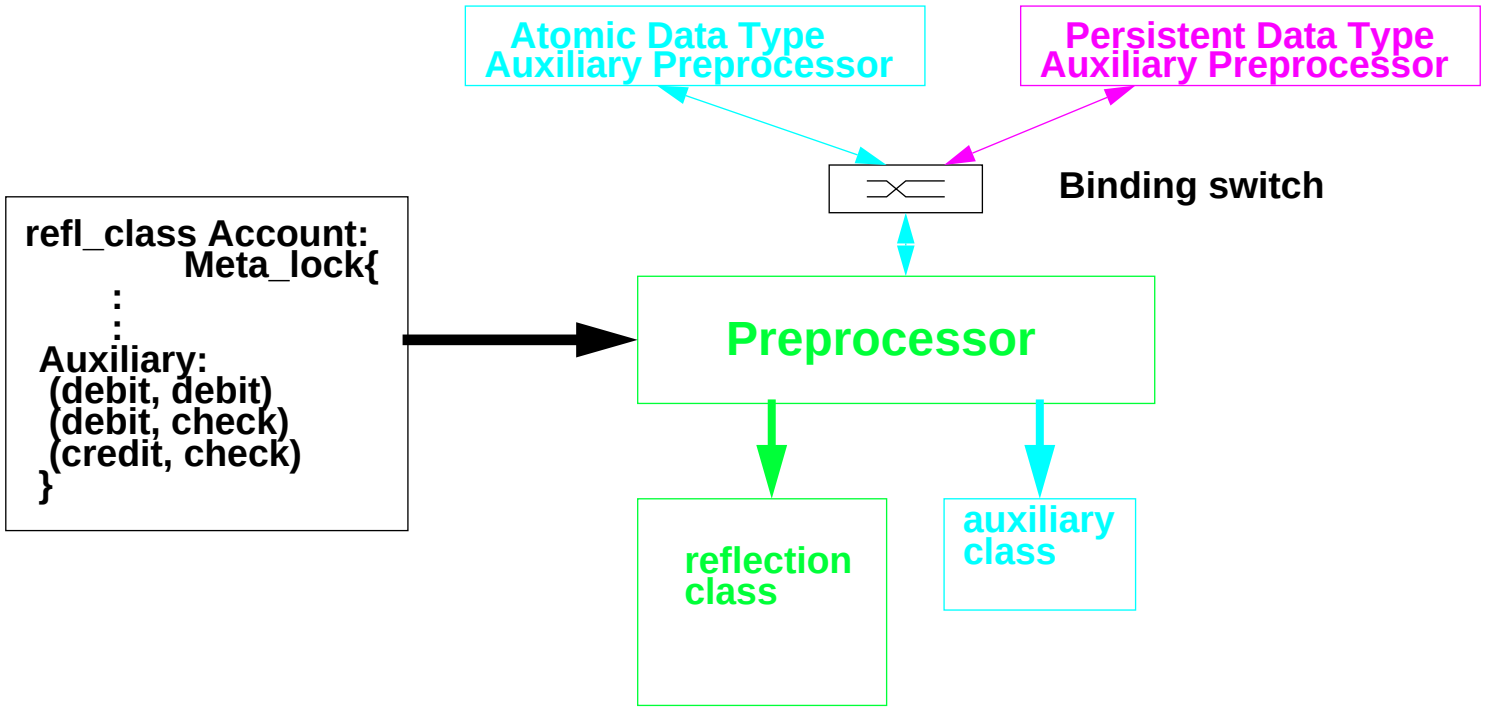
```
class Account {

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

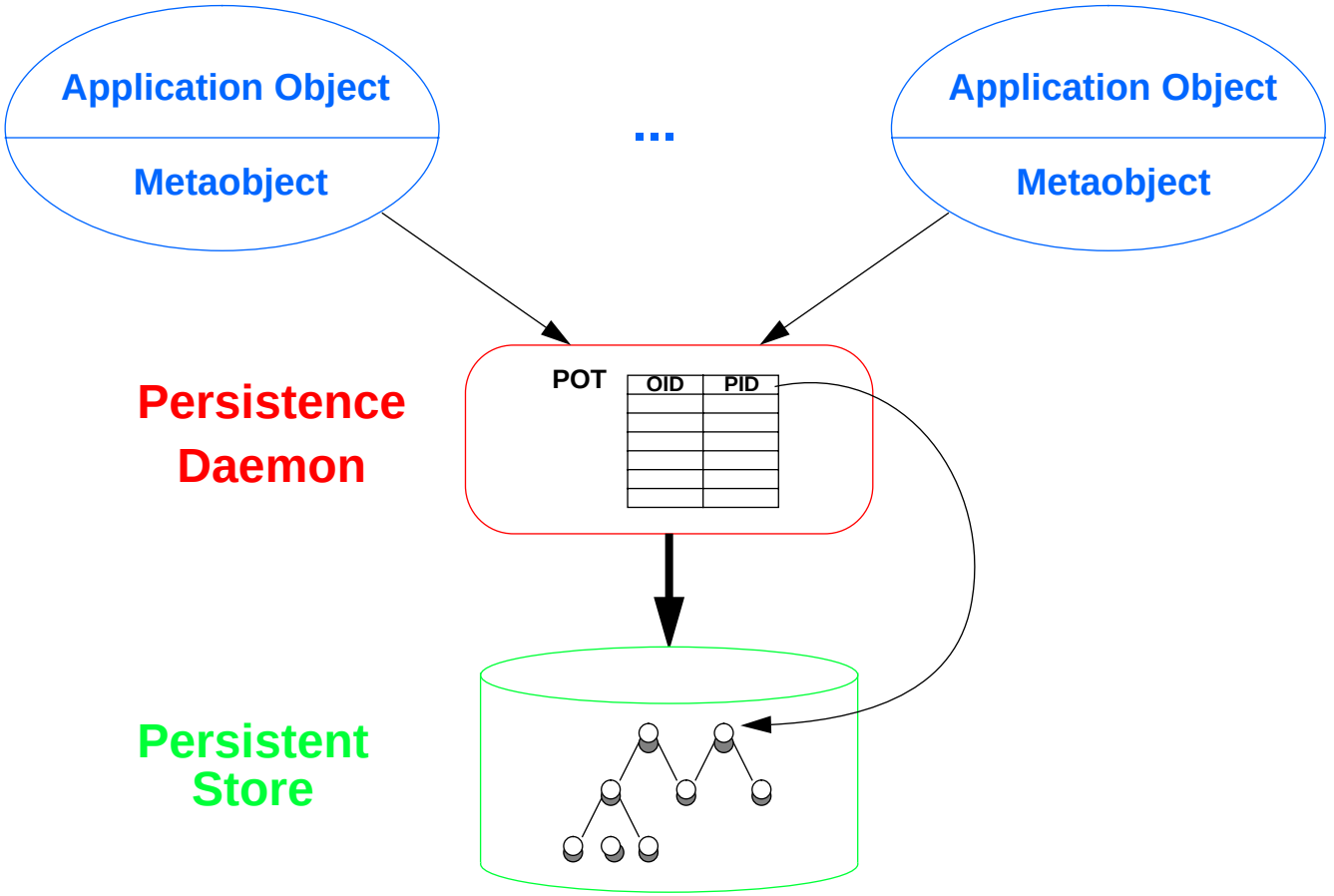
```
class Meta_Lock
          extends MetaObject {

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();
}
```

Auxiliary Information

- Preprocessor can be customised through auxiliary preprocessors
- Auxiliary information is processed by auxiliary preprocessor
- Auxiliary class is passed to the metaobject when it is created

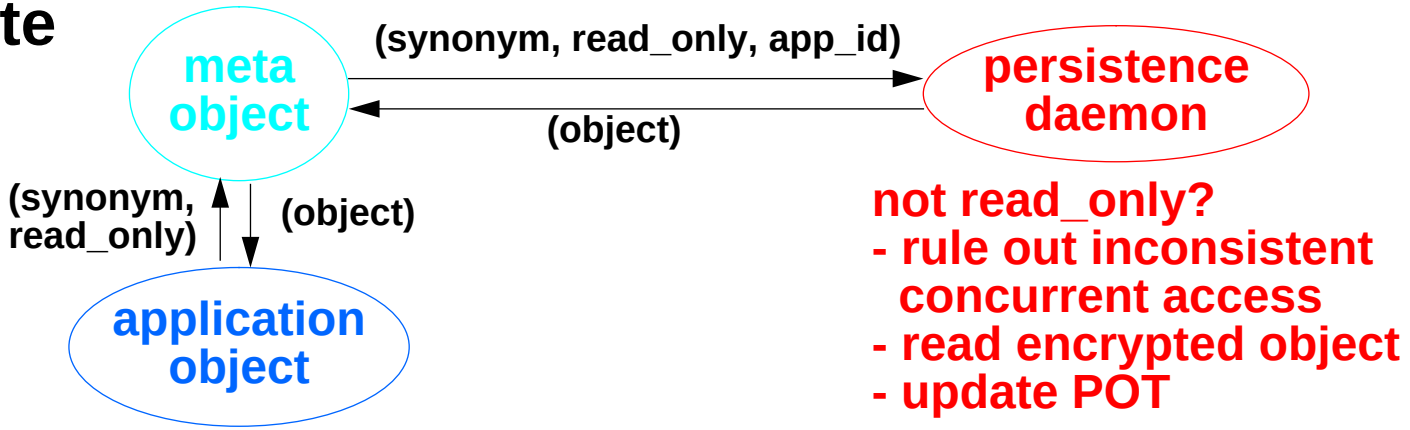


A Persistence Service for Java



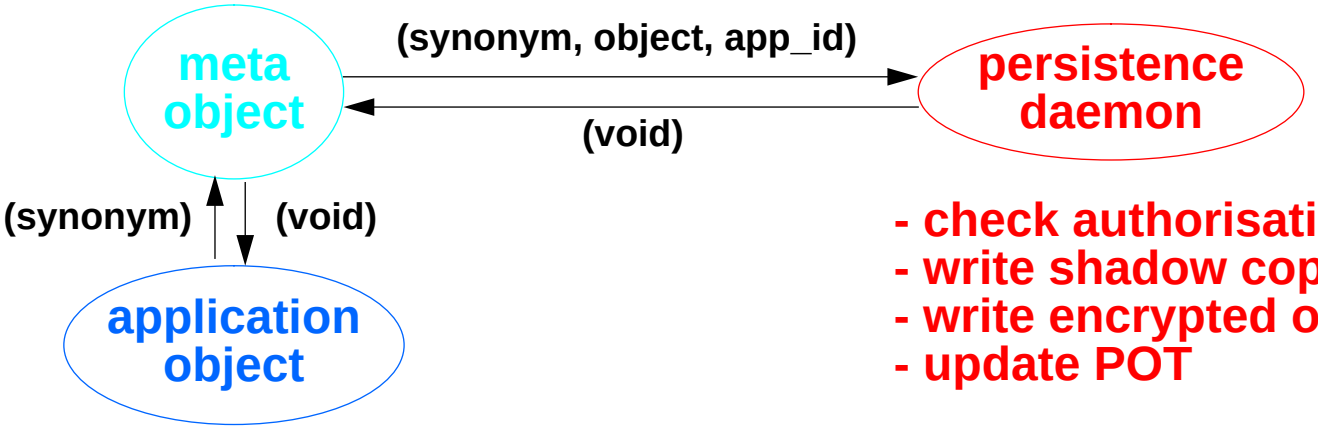
Activation and Deactivation

Activate



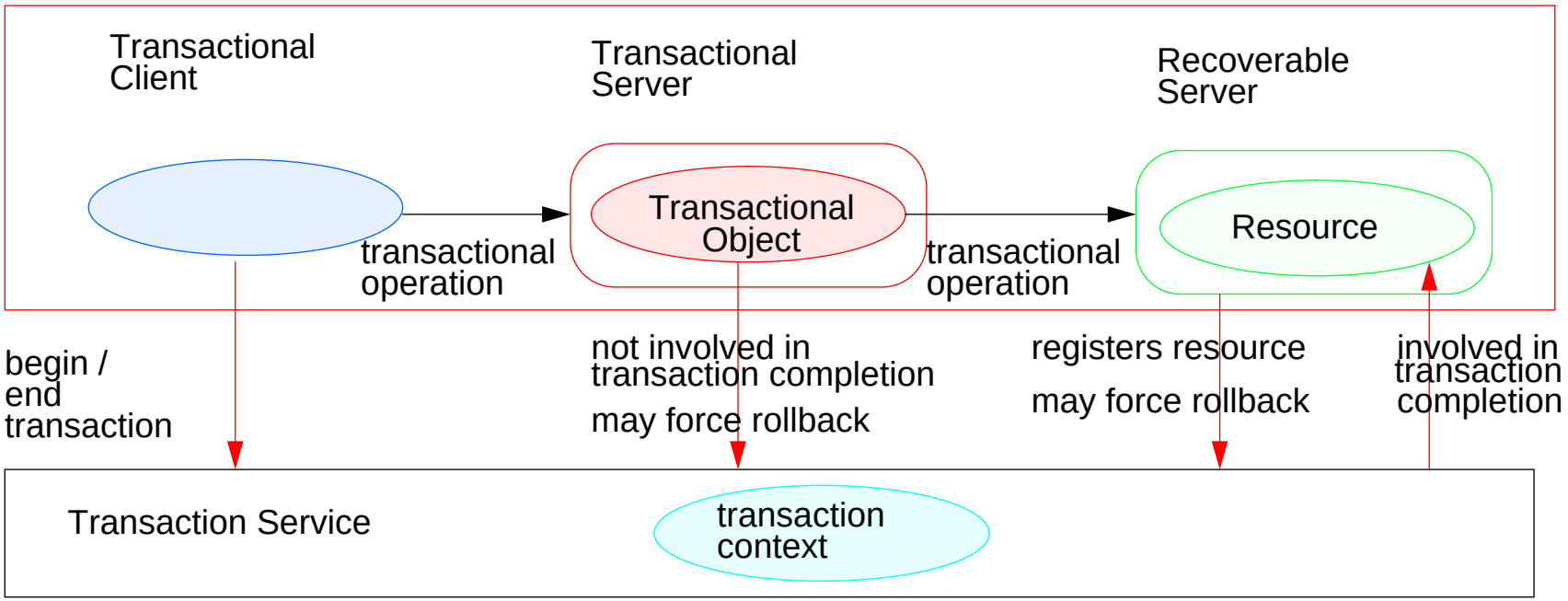
not read_only?
- rule out inconsistent concurrent access
- read encrypted object
- update POT

Deactivate



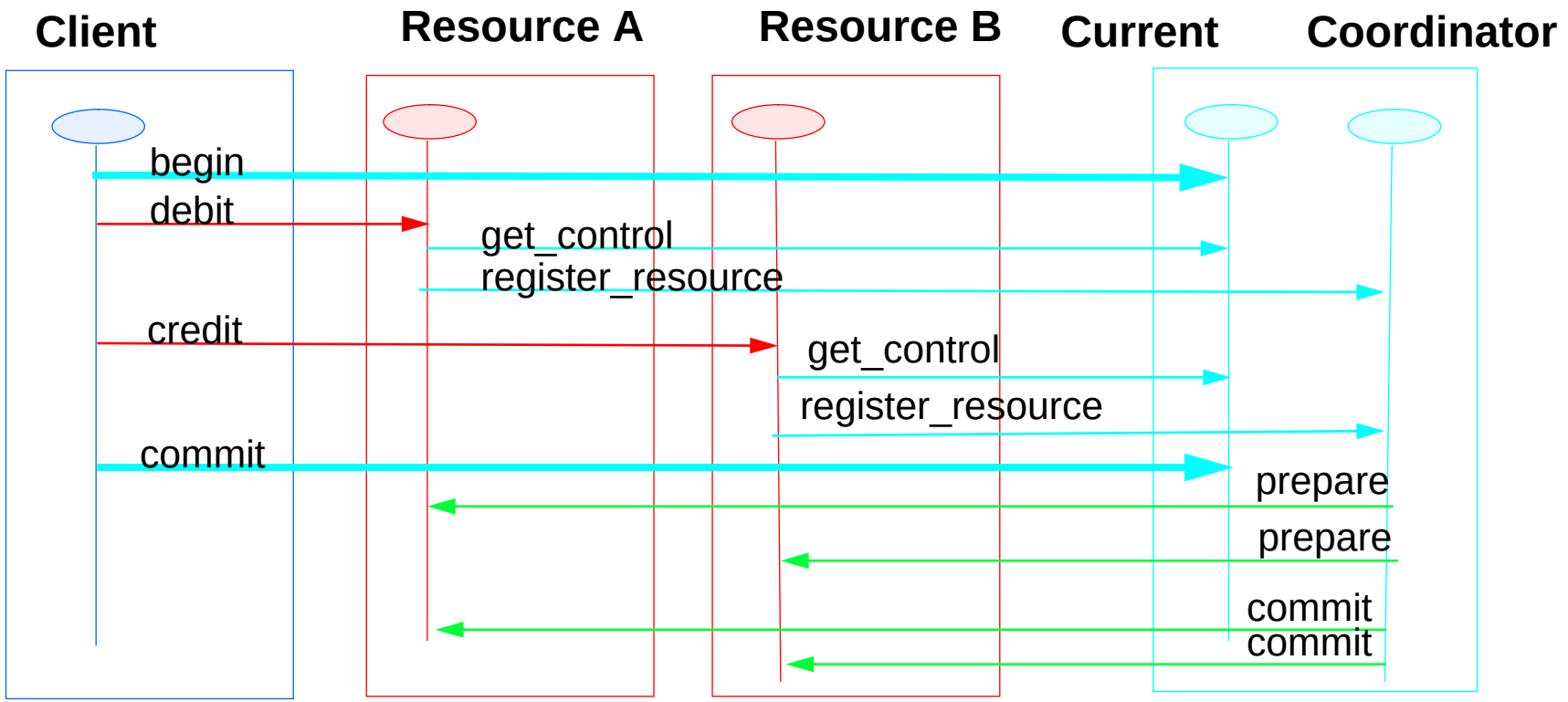
- check authorisation
- write shadow copy
- write encrypted object
- update POT

Object Transaction Service



A Transaction Example

- Every object method: deals with concurrency, and registers itself
- Every object: provides prepare, commit, and rollback operations



Recoverable Object

- **An object method**
  - **uses the Control parameter to retrieve the Coordinator object**
  - **registers itself to the transaction service via the Coordinator**
  - **checks whether it is registered for the same transaction**
  - **ensures it is involved only in one transaction at a time**
- **The above is concerned with non-functional requirements**
- **Functional and non-functional code are mixed up**

```
class Account  extends Resource implements Transactional  {
 public void credit( Control ctl, double amt) {
   Coordinator co = ctl.get_coordinator( );
   //make sure this object has not been registered for the same transaction
   //make sure this object is involved only in one transaction at a time
    RecoveryCoordinator r = co.register_resource(this);
   balance = balance + amt;
 }
 public Vote prepare(...) { .....};
 public void commit(...) {.....};
 public void rollback(...) {.....};
}
```

Recoverable Object (Using MOP)

- **Only functional requirements are implemented in application objects**
- **Non-functional requirements are implemented in metaobjects**
- **Multiple concurrency control methods can be provided**
- **Users can choose a method suitable for their particular application either statically or dynamically**

```
class meta_2pl extends MetaObject {
  public void metaBefore(MID mid, CID cid, Arg args)
   {
     Control ctrl = (Control) args.extractArg(0).extractObject( );
     Coordinator co = ctl.get_coordinator( );
     //make sure this object has not been registered for the same transaction
     //make sure this object is involved in only one transaction at a time
      RecoveryCoordinator r = co ->register_resource(this);
   }
}
```

```
  class Account extends Resource {
   public void credit( Control ctl,    double amt)
    {   balance = balance + amt; }
  }
```

Benefits

- Flexibility to customise policies dynamically to suit run-time environment
- Binding specification can be changed independently from object implementation
- Based on a simple, powerful and popular language
- Support for providing auxiliary information to metaobjects

Information

- First version is available to ANSA sponsors
- Second version will focus on improving performance
- Technical contacts:
  - Zhixue Wu: zw@ansa.co.uk 01223--568930
  - Scarlet Schwiderski: ss@ansa.co.uk
- Product contacts:
  - Carrie Story: carrie@ansa.co.uk 01223--568926
  - Billy Gibson: wag@anda.co.uk 01223--568940