The Problem

• aspectizing Concurrency and Failures
• AspectJ & Transactions
• aspectize transaction semantics
• aspectize transaction interfaces
• aspectize transaction mechanisms
Overview

1. Transactions
2. AOP (Aspect Oriented Programming)
3. aspectize transaction semantics
4. aspectize transaction interfaces
5. aspectize transaction mechanisms
6. Conclusion

Transactions

- ACID
- Serializability
- Concurrency
- Failure Recovery

=> fundamental paradigm to handle Concurrency and Failures

AOP

- separation of concerns
  - separate code (aspect)
- Weaving
  \[\text{Code + Aspect(s) = Program}\]
AOP: Example

- [Picture: Figure/Point/Line Example]

AOP: JoinPoint

- JoinPoint:
  - point in flow of program
  - „hook“ to insert aspect-code

```
pointcut moves():
    receptions(void Line.setP1(Point)) ||
    receptions(void Line.setP2(Point)) ||
    receptions(void Point.setX(int)) ||
    receptions(void Point.setY(int));
```
AOP: Advice

- Before
- After
- Around

AOP: Example

```java
aspect MoveTracking {
    static boolean flag = false;
    static boolean testAndClear() {
        boolean result = flag;
        flag = false;
        return result;
    }
    pointcut moves() : ... after() : moves() {
        flag = true;
    }
}
```

Aspectize Transaction Semantics

- combine non-transactional code with transaction aspects
- problems:
  - irreversible actions
  - deadlock
ATS: Unsolvable Deadlock

T1:
A.deposit(Amount) while (B.getBalance() <= Amount) {} B.withdraw(Amount)

T2:
B.deposit(Amount) while (A.getBalance() <= Amount) {} A.withdraw(Amount)

- Problem: linearizability ≠ serializability
- Solution: multithreaded transactions
- New Problem: automatic detection of cooperating threads

Aspectize Transaction Interfaces

Aspectize Transaction Mechanisms
Conclusion

- aspectize transaction semantics: not possible
- aspectize transaction interfaces: possible, but artificial
- aspectize transaction mechanisms: only syntactical separation