



CS544 EA

Applications

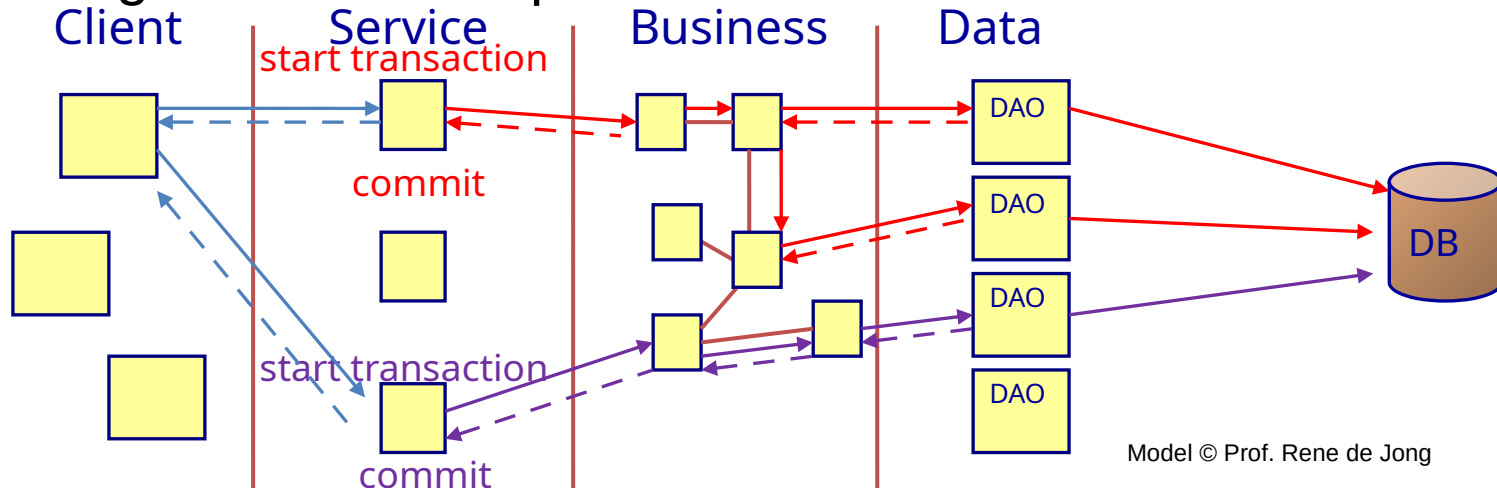
Spring Transactions

Spring Transaction Support

- Spring is not a transaction manager
 - We still need a transaction manager
 - JDBC transaction manager
 - Hibernate transaction manager
 - XA transaction manager (JTA)
- Spring provides an **abstraction for TX management**
 - You declare how transactions should be managed
 - Spring make it work with the underlying transaction manager

Transaction Demarcation

- The transactional demarcation is the specification of the **transactional boundaries**
- This is typical at the service level
 - Multiple DAO's can be involved in one transaction
 - Creating a transaction per unit of work



BMT

```
public class CustomerService {  
    private CustomerDAO customerDao = new CustomerDAO();  
    private AddressDAO addressDao = new AddressDAO();  
    private CreditCardDAO ccDao = new CreditCardDAO();  
    private EntityManager em = EntityManagerHelper.getCurrent();  
  
    public void addNewCustomer(Customer cust, Address shipAddr, CreditCard cc,  
                               Address billAddr) {  
        cc.setAddress(billAddr);  
        cust.setShipAddress(shipAddr);  
        cust.setCreditCard(cc);  
  
        em.getTransaction().begin();  
        addressDao.create(shipAddr);  
        addressDao.create(billAddr);  
        ccDao.create(cc);  
        customerDao.create(cust);  
        em.getTransaction().commit();  
    }  
}
```

Programmatically begins
the transaction

Transaction is automatically
propagated to enclosed methods

Programmatically ends
the transaction

...

CMT

@Service

```
public class CustomerService {  
    private CustomerDAO customerDao;  
    private AddressDAO addressDao;  
    private CreditCardDAO ccDao;
```

Simply declare that a transaction is needed

REQUIRED is the default and therefore optional

```
@Transactional(propagation=Propagation.REQUIRED)
```

```
public void addNewCustomer(Customer cust, Address shipAddr, CreditCard cc,  
    Address billAddr) {
```

```
    cc.setAddress(billAddr);  
    cust.setShipAddress(shipAddr);  
    cust.setCreditCard(cc);
```

Spring takes care of opening and closing the TX

```
    addressDao.create(shipAddr);  
    addressDao.create(billAddr);  
    ccDao.create(cc);  
    customerDao.create(cust);
```

Transaction propagates to called methods as normal

```
}
```

...

Class Annotations

```
@Repository
@Transactional(propagation = Propagation.REQUIRED)
public class AddressDao {
```

Annotating a class specifies that all its methods should be Transactional

```
    @PersistenceContext
    private EntityManager em;
```

```
    @Transactional(propagation = Propagation.MANDATORY)
    public void create(Address addr) {
        em.persist(addr);
    }
```

You can add method level annotations to specify exceptions

```
    public Address get(int id) {
        return em.find(Address.class, id);
    }
```

```
    public void update(Address addr) {
        em.merge(addr);
    }
```

```
    public void delete(Address addr) {
        em.remove(addr);
    }
```

These are propagation REQUIRED

```
}
```

Additional Options

- You can also specify the **isolation** level

```
@Repository
@Transactional(propagation = Propagation.REQUIRED, isolation=Isolation.READ_COMMITTED)
public class AddressDao {

    @PersistenceContext
    private EntityManager em;
```

- Or that a transaction should be **read only**

```
@Repository
@Transactional
public class AddressDao {

    @Transactional(readOnly=true)
    public Address get(int id) {
        return em.find(Address.class, id);
    }
}
```

Additional Options

- A **timeout** in seconds (needs TXManager support)

```
@Repository
@Transactional
public class AddressDao {

    @Transactional(timeout=10)
    public void update(Address addr) {
        em.merge(addr);
    }
}
```

By default rollback for unchecked exceptions
but not for checked exceptions

- What exceptions to **rollback** for

```
@Repository
@Transactional(
    rollbackFor={MyCheckedException.class},
    noRollbackFor={MyRuntimeException.class}
)
public class AddressDao {
```