Codebeispiel 1 Interface für TransactionParticipant

```
public interface TransactionParticipantIF {

    public void startTransaction();
    public void commit();
    public void abort();

}
```

Codebeispiel 2 Datenbasis für Transaktion

```
class Data {

    int getValue();
    void setValue(int newValue);
    private int value = 0;

}
```

Lost Update

Transaktion_1	Transaktion_2
int x = a.getValue()	
x+=20	
	int y = a.getValue()
	y+=100
a.setValue(x)	
	a.setValue(y)

Dirty Read

Transaktion_1	Transaktion_2
int x = a.getValue()	
x+=300	
a.setValue(x)	
	int y = a.getValue()
	y*=2
	a.setValue(y)
abort()	

Unrepeatable Read

Transaktion_1	Transaktion_2
int x = a.getValue()	
	int y = a.getValue()
	y+=1
	a.setValue(y)
int z = a.getValue()	
if (x==z) ...	

Codebeispiel 3 Zustandssicherung mittels Objektcloning

```
class TransactionManager {
    TransactionParticipant p1, p2, ...;
    // zum Backup der TP's
    Object b1, b2, ...;
    startTransaction() {
        b1 = p1.clone();
        b2 = p2.clone();
        ...
    }
    commit() {
        b1 = null;
        b2 = null;
        ...
    }
    abort() {
        p1 = (TransactionParticipant)b1;
        p2 = (TransactionParticipant)b2;
        ...
    }
}
```

Codebeispiel 4 Zustandssicherung mittels Objektserialisierung

```
class TransactionManager {
    TransactionParticipant p1, p2, ...;
    ObjectOutputStream out;
    ObjectInputStream in;
    startTransaction() {
        out.writeObject(p1);
        out.writeObject(p2);
        ...
    }
    commit() {
        // gespeicherte Objekte verwerfen
        out.discard();
    }
    abort() {
        // Objekte rekonstruieren
        p1 = (TransactionParticipant)in.readObject();
        p2 = (TransactionParticipant)in.readObject();
        ...
    }
}
```

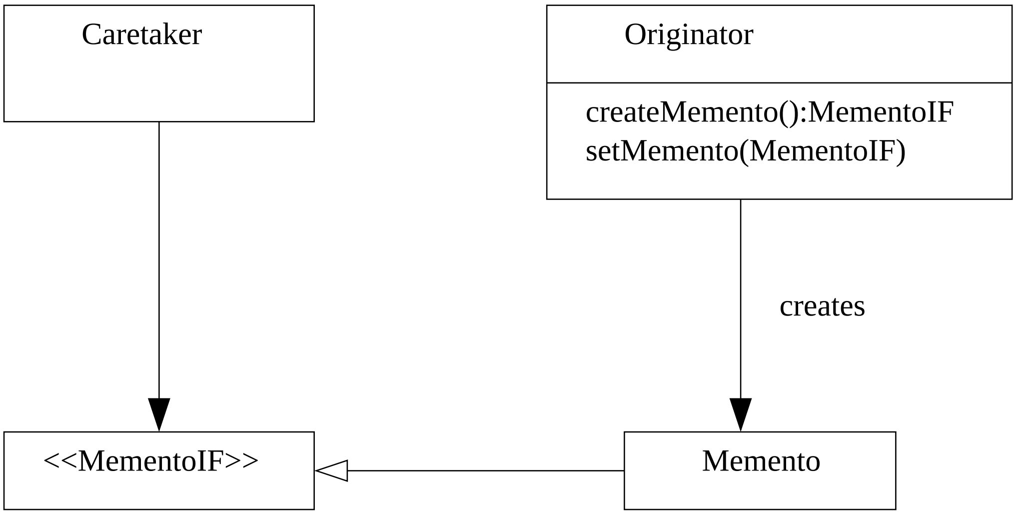

Abbildung 1: Klassendiagramm des Snapshot-Pattern

Codebeispiel 5 Zustandssicherung mittels Snapshot-Pattern

```
TransactionManager {
    TransactionParticipant d1, d2, ...;
    MementoIF b1, b2,  ...;
    startTransaction() {
        b1 = d1.createMemento();
        b2 = d2.createMemento();
        ...
    }
    commit() {
        b1 = null;
        b2 = null;
    }
    abort() {
        d1.setMemento(b1);
        d2.setMemento(b2);
    }
}
```

Codebeispiel 6 TransactionManager benutzt gepufferte Objekte

```
TransactionManager {
    TransactionParticipantIF p1, p2;
    startTransaction() {
        p1.startTransaction();
        p2.startTransaction();
    }
    commit() {
        p1.commit();
        p2.commit();
    }
    abort() {
        p1.abort();
        p2.abort();
    }
}
```

Codebeispiel 7 TransactionParticipant mit Änderungspuffer

```
public class DataClass
implements TransactionParticipantIF {
    private int value;
    private DataClassBuffer buffer;
    public int getValue() {
        if (buffer != null)
            return buffer.getValue();
        else
            return value;
    }
    public void setValue(int value) {
        if (buffer != null)
            buffer.setValue(value)
        else
            this.value = value;
    }
    public void startTransaction(){
        if (buffer == null) {
            buffer = new DataClassBuffer(this);
        } else {
            // bin schon in einer transaktion
        }
    }
    public void abort() {
        buffer = null;
    }
    public void commit() {
        this.value = buffer.getValue();
        buffer = null;
    }
    // inner class to enforce
    // strong encapsulation
    class DataClassBuffer {
        int value;
        private DataClassBuffer() {};
        DataClassBuffer(DataClass c) {
            this.value = c.getValue();
        }
        void setValue(int value) {
            this.value = value;
        }
        int getValue() {
            return value;
        }
    }
}
```

Codebeispiel 8 Accessor Methode liefert neues Objekt zurück

```
class CalendarEvent {
    Date start;
    Date end;
    public Date getStart() {
        return start;
    }
    ...
}
```

Diese Klasse wird nun benutzt

```
CalendarEvent event;
Date d = event.getStart();
d.modify();
```

Diese letzte Anweisung verändert auch `event.start` !
Daher schreiben wir die Datenklasse um

```
class CalendarEvent2 {
    ...
    public getStart() {
        return new Date(start.getTime());
    }
}
```

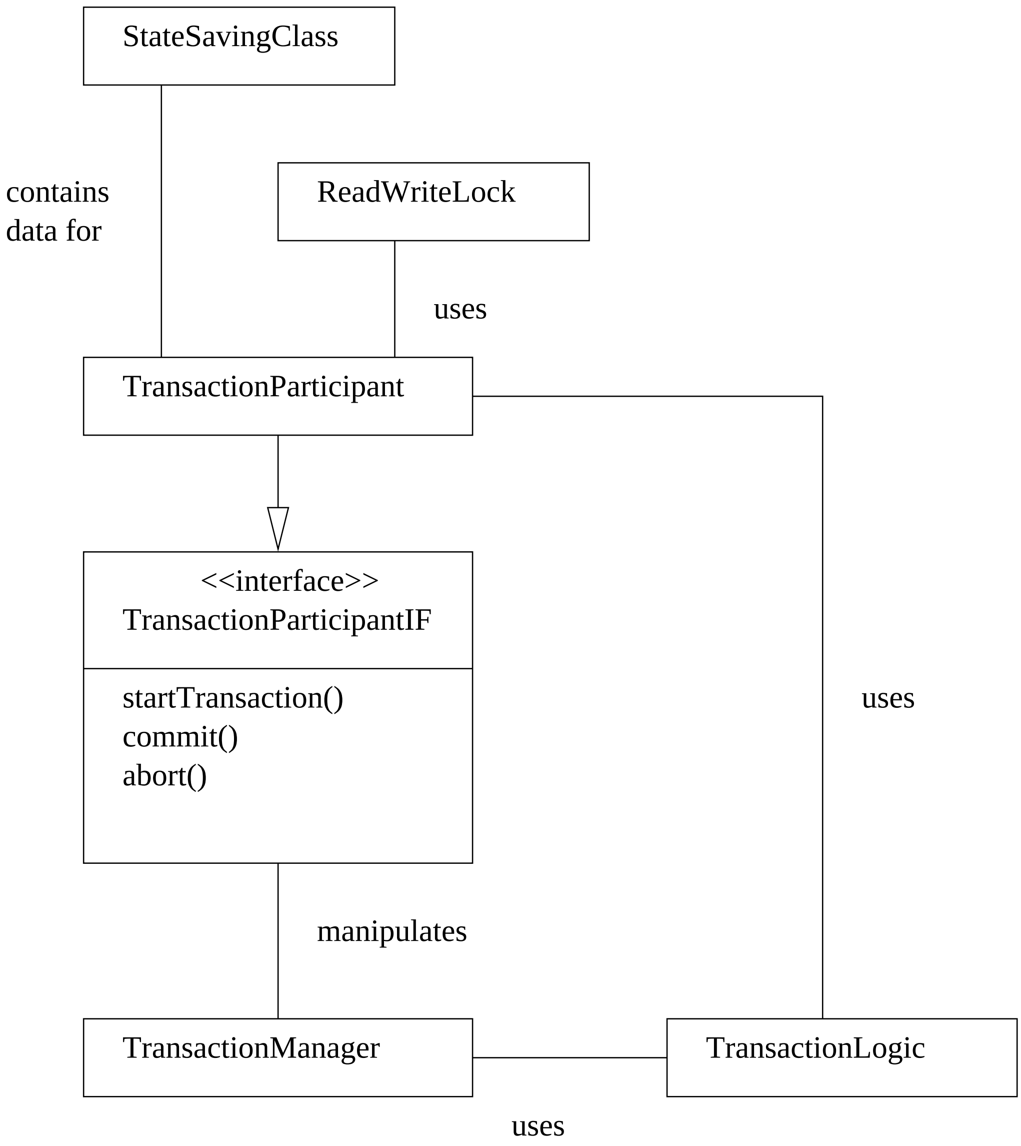

Abbildung 2: generische ACID Transaktion

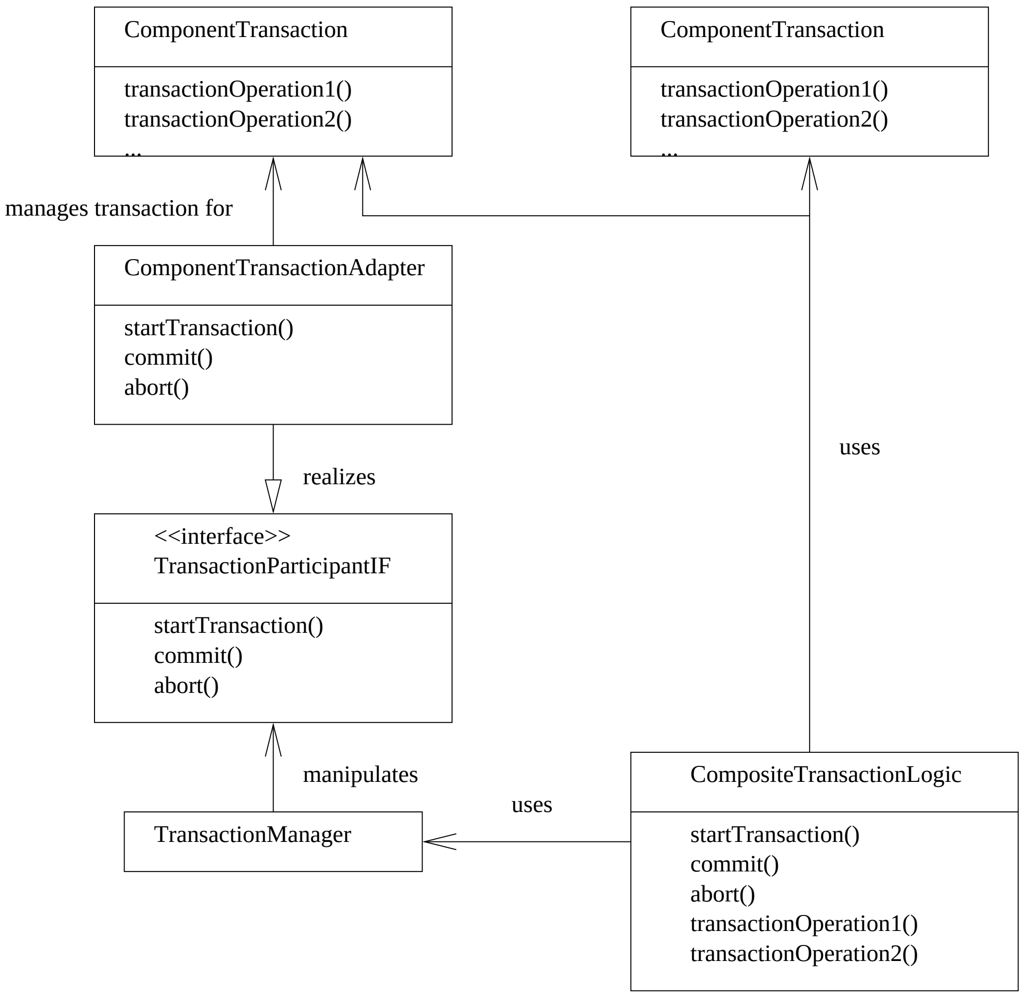

Abbildung 3: composite Transaction Pattern

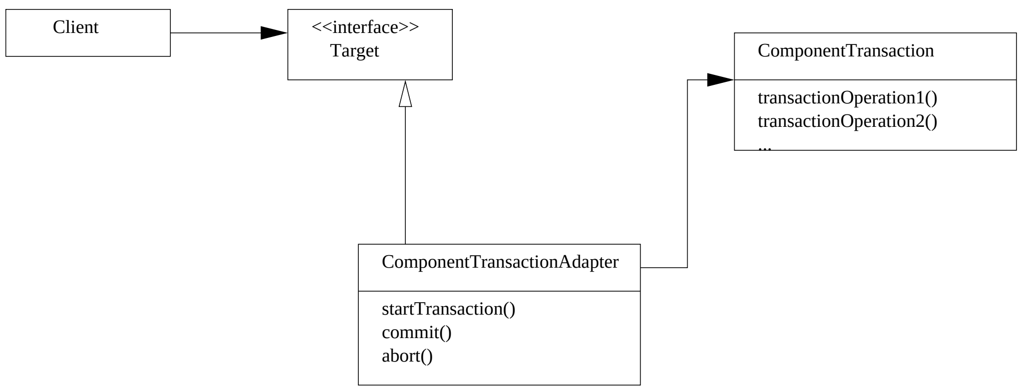

Abbildung 4: Adapter Pattern Beispiel

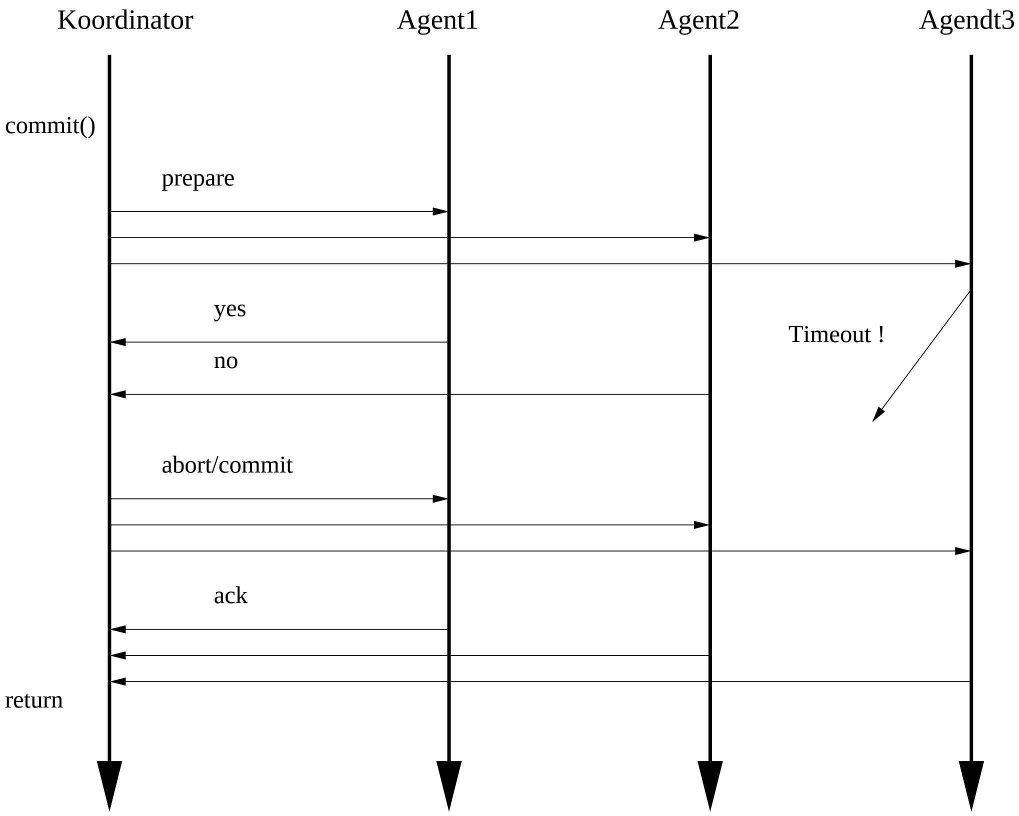

Abbildung 5: Ablauf des 2PC Protokoll

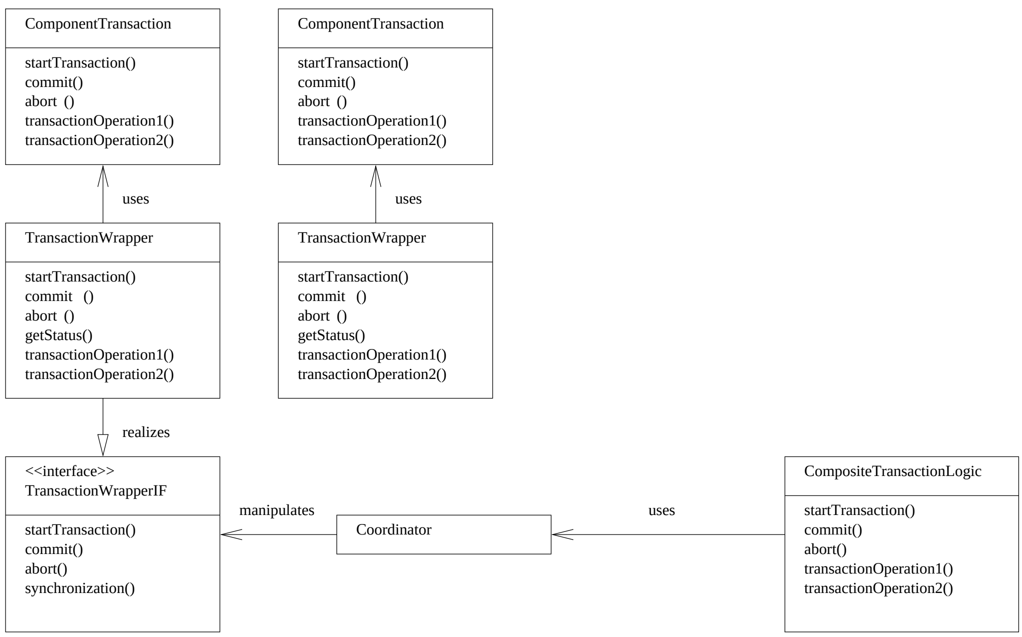

Abbildung 6: Klassendiagramm des 2PC Pattern

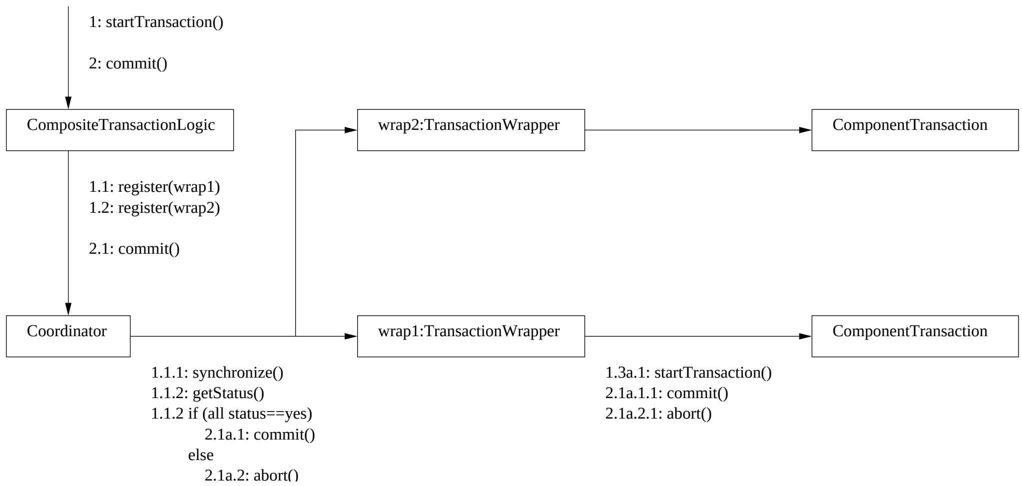

Abbildung 7: Ablaufdiagramm des 2PC

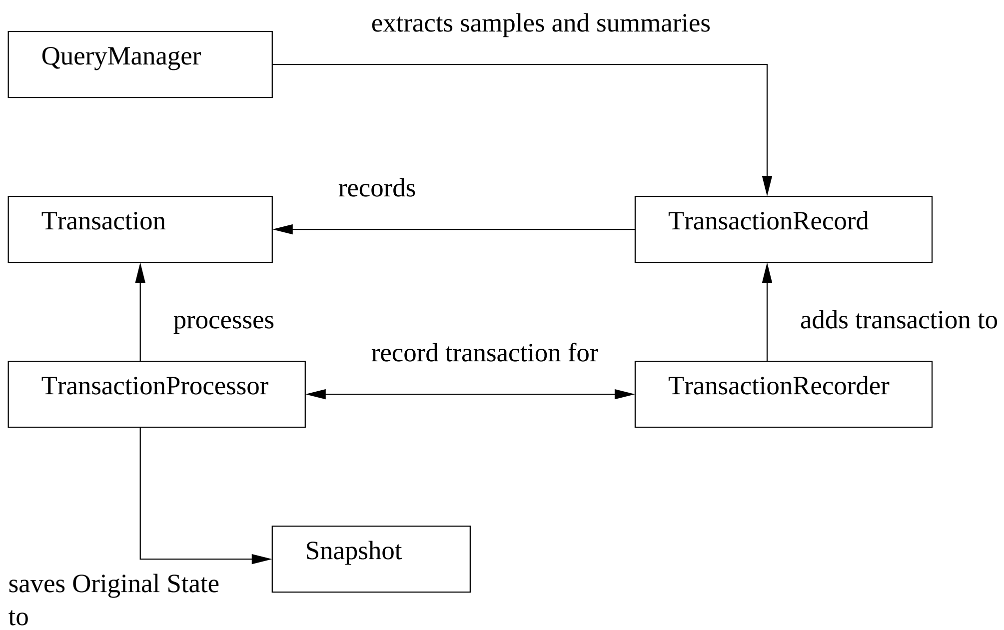

Abbildung 8: Klassendiagramm des Audit Trail Pattern