Was der Entwickler über Transaktionen in verteilten Systemen wissen muss

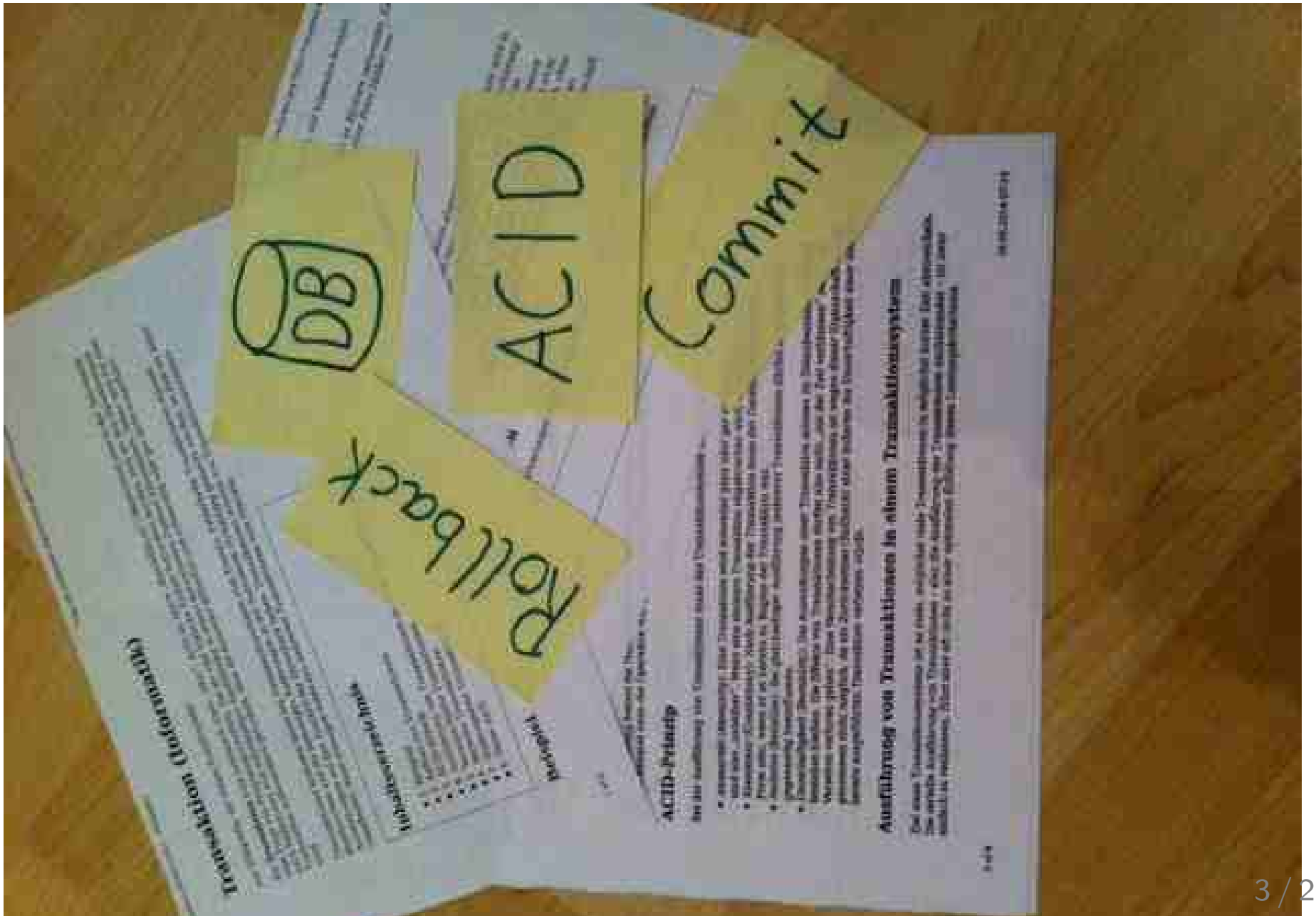
modellierung.net

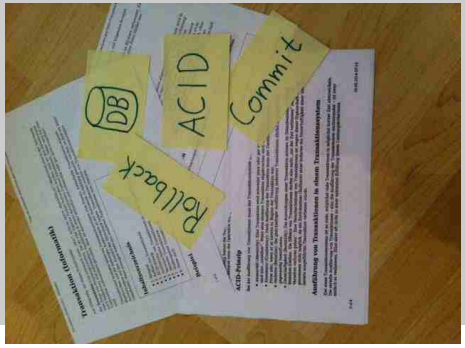
15. Juli 2014

Inhaltsverzeichnis

Hauptthema: Wissenswertes über Transaktionen im Allgemeinen und JTA im Speziellen

Definition des Zwecks





Bitte versuchen, den Zweck von Transaktionen in eigene Worte zu fassen

Zweck Teil 1:

Systeme bestehen aus Invarianten (Prädikaten)

Bekanntes Beispiel: Die Summe aller Konten muss 0 ergeben

Prädikate am konkreten Beispiel

Prädikate unseres Systems:

- TransaktionsId für Vendor ist *unique*
- Anzahl waiting Jobs = Anzahl ausstehender Nachrichten
- Anzahl Teilnehmer = Anzahl erzeugte + Anzahl migrierte Teilnehmer

Prädikate am konkreten Beispiel

Prädikate unseres Systems:

- TransaktionsId für Vendor ist *unique*
- Anzahl waiting Jobs = Anzahl ausstehender Nachrichten
- Anzahl Teilnehmer = Anzahl erzeugte + Anzahl migrierte Teilnehmer

Prädikate am konkreten Beispiel

Prädikate unseres Systems:

- TransaktionsId für Vendor ist *unique*
- Anzahl waiting Jobs = Anzahl ausstehender Nachrichten
- Anzahl Teilnehmer = Anzahl erzeugte + Anzahl migrierte Teilnehmer

Prädikate am konkreten Beispiel

Prädikate unseres Systems:

- TransaktionsId für Vendor ist *unique*
- Anzahl waiting Jobs = Anzahl ausstehender Nachrichten
- Anzahl Teilnehmer = Anzahl erzeugte + Anzahl migrierte Teilnehmer

Zweck Teil 2:

Solange der ausgeführte Code diese Prädikate als Postcondition nicht verletzt, garantiert die Transaktion, dass die Invarianten stets gelten.

Da helfen auch TX nicht

Ziehe von A 100 ab, addiere 101 zu B

Zweck Teil 2:

Solange der ausgeführte Code diese Prädikate als Postcondition nicht verletzt, garantiert die Transaktion, dass die Invarianten stets gelten.

Da helfen auch TX nicht

Ziehe von A 100 ab, addiere 101 zu B

Trotz Stromausfall, Netzwerkstörung etc. garantieren Transaktionen die Einhaltung der Prädikate. Diese Definition gilt für verteilte Systeme und weicht ggf. von Wikipedia und anderen Quellen, die Transaktionen teilweise nur als Datenbank-TX sehen, ab.

JEE

Nathaniel Borenstein

The most likely way for the world to be destroyed, most experts agree, is by accident. That's where we come in: we're computer professionals. We cause accidents.

Ressourcen

Transaktional:

- Datenbankoperationen
- Message senden
- Message verarbeiten
- EJB-Calls[1]

[1] mit Ausnahmen

Ressourcen

Nicht transaktional:

- Kommunikation über Sockets
- Dateien schreiben
- Logging
- Web-Services[2]

[2] Ausnahme: WS-Transaction

Abschluss von Transaktionen

Betrachtung Container Managed Transactions (CMT): Implementierung im Container

- Interceptoren/Proxies
- AOP
- generierter Code

Bei Interceptoren hat der Container keine Möglichkeit, plain-Java calls mit Transaktionen zu versehen. U.a. deshalb sind laut Spezifikation nur Aufrufe über ein EJB-Interface transaktional. An POJOs und CDI-Beans hat die TransactionAttribute-Annotation keinen Effekt.

Start von Transaktionen

Bedingungen für Start:

- EJB-Call
- Transaktionsattribut REQUIRED oder REQUIRES_NEW
- Bei REQUIRED: keine TX des Aufrufers

Abschluss von Transaktionen

Rollback:

- setRollbackOnly
- RuntimeException[3] über EJB-Call-Grenze
- Checked Exception[4] über EJB-Call-Grenze

[3] die nicht mit ApplicationException.rollback markiert wurde

[4] die mit ApplicationException.rollback markiert wurde

Abschluss von Transaktionen

Commit:

- EJB-Call (der TX gestartet hat) endet
- checked Exception[5]

[5] und nicht mit ApplicationException.rollback annotiert

EJB-Call

Aufrufe über:

- Remote- + Local-Interface
- Injectete Referenz (auch @LocalBean)
- SessionContext.getBusinessObject

Aufrufe über:

- Remote- + Local-Interface
- Injectete Referenz (auch @LocalBean)
- SessionContext.getBusinessObject

SessionContext.getBusinessObject bietet die Möglichkeit, aus einer Methode einer EJB eine Methode desselben Objektes über einen EJB-Call aufzurufen.

Neue Transaktion bei this-Aufruf

```
@javax.ejb.Stateless
@javax.ejb.Local(SomeInterface.class)
public class SomeInterfaceImpl implements SomeInterface {
   @javax.annotation.Resource
   private SessionContext sessionContext;

   @Override //has the TX-Attribute REQUIRED
   public List<String> doSomething(){
      sessionContext.getBusinessObject(SomeInterface.class).doInNewTx();
      throw new RuntimeException(); //Abort the Transaction
   }

   @Override
   @javax.ejb.TransactionAttribute(REQUIRES_NEW)
   public void doInNewTx(){
      /* persist something with the EntityManager, it will be persisted
         although the caller rolls back its Tx, cause this method runs
         in its own transaction */
   }
}
```

Mehrere Ressourcen

Commit auf mehreren Ressourcen?

- 2-Phase-Commit
- XA-Transaktionen
- Last-Resource

XA Transaktionen

XAException

- Communication failure
- Deadlock
- Protocol Error
- Timeout
- heuristically completed / committed / rolled back
- ...

XAException
- Communication failure
- Deadlock
- Protocol Error
- Timeout
- heuristically completed / committed / rolled back
- ...

javax.transaction.xa.XAException als Beispiel für Probleme bei verteilten TX. Zwar können einige intern vom ResourceManager gelöst werden, trotzdem werden XA-Transaktionen nur verwendet, wo es nicht vermeidbar ist. Falls die Kommunikation zwischen den an der TX beteiligten Knoten zusammenbricht, müssen Locks gehalten werden, bis Kommunikation wieder möglich ist (weil ggf. einige Beteiligte bereits committed haben).

Langlaufende Businesstransaktionen

Hört sich gut an! Aber:

- langes Locking
- inkompatible Protokolle (EDIFACT auf FTP-Server)
- technische Kommunikationsprobleme

Hört sich gut an! Aber:

- langes Locking
- inkompatible Protokolle (EDIFACT auf FTP-Server)
- technische Kommunikationsprobleme

Wer eine gute Idee für langlaufende Businesstransaktionen hat: Firma gründen und die Lösung als Produkt verkaufen.

Verteilte Transaktionen ohne XA

- Kompensation
- Idempotenz
- Beachten: Lock-Zeiten!
- JMS

Zusammenfassung

- Zweck einer Transaktion
- JEE
- Transaktionen in Unternehmensprozessen

- Prädikate garantieren
- CMT, wann wird Tx abgeschlossen
- keine langlaufenden Tx, Idempotenz, Kompensation und Entkoppelung (z.B. JMS)