

**DezSys-Protokoll**

**DezSysLabor-05 "Verteilte Transaktionen"**

**Dezentrale Systeme**

**5BHITT 2015/16**

**Thomas Stedronsky**

|  |  |
| --- | --- |
|  | **Version 1.0** |
| **Note:** | **Begonnen am 15. Jänner 2016** |
| **Betreuer: Th. Micheler** | **Beendet** |

Inhaltsverzeichnis

[1 Einführung 3](#_Toc441136764)

[1.1 Ziele 3](#_Toc441136765)

[1.2 Voraussetzungen 3](#_Toc441136766)

[1.3 Aufgabenstellung 3](#_Toc441136767)

[2 Ergebnisse 5](#_Toc441136768)

[3 Quellen 6](#_Toc441136769)

[4 Abbildungsverzeichnis 7](#_Toc441136770)

# Einführung

Die Übung soll die Grundlagen von verteilte Transaktionen mit Hilfe eines praktischen Beispiels in JAVA vertiefen.

## Ziele

Implementieren Sie in JAVA einen Transaktionsmanager, der Befehle an mehrer Stationen weitergibt und diese koordiniert. Mit Hilfe des 2-Phase-Commit Protokolls sollen die Transaktionen und die Antwort der Stationen verwaltet werden. Der Befehl kann beliebig gewaehlt werden und soll eine Datenquelle (Datenbank oder Datei oder Message Queue) abfragen oder veändern.

Die Kommunikation zwischen Transaktionsmanagers und der Stationen soll mit Hilfe einer Übertragungsmethode (IPC, RPC, Java RMI, JMS, etc) aus dem letzten Schuljahr umgesetzt werden.

## Voraussetzungen

* Grundlagen Transaktionen (allgmein, Datenbanksysteme)
* Anbindung Datenquelle in JAVA (JDBC, File, JMS)
* Kommunikation in JAVA (IPC, RPC, Java RMI, JMS)

## Aufgabenstellung

Der Transaktionsmanager läuft auf einer eigenen Instanz (bzw. eigenem Port) und stellt die Schnittstelle zwischen den Stationen und dem Benutzer dar. Der Benutzer gibt über die Konsole oder ein User Interface einen Befehl ein, der danach an alle Stationen verteilt wird. Da das 2-Phase-Commit Protokoll als Transaktionsprotokoll zugrunde liegt, soll der Transaktionsmanager jeweils nach einem Befehl,

* das Resultat nach der PREPARE-Phase (Bsp. 3xYES 0xNO 0xTIMEOUT) ausgeben
* welche Aktion der Transaktionsmanager danach durchfuehrt (doCommit, doAbort)
* das Resultat der COMMIT-Phase (Bsp. 3xACK 0xNCK 0xTIMEOUT)
* danach kann ein neuer Befehl eingegeben werden

Logging:  
Um im Einzelfall die Transaktionen und Resultat nachverfolgen zu koennen, sollen alle Befehle und deren Resultate mit Timestamp geloggt werden. Beim Transaktionsmanager soll dokumentiert werden, welcher Befehl zu welcher Station und zu welchem Zeitpunkt abgesendet wurde, ebenso beim Erhalt der Antwort. Ebenso sollen, bei den Stationen eingehenden Befehle und deren Resultate bei Ausführung an der lokalen Datenquelle mitdokumentiert werden.

Die folgende Grafik soll den Vorgang beim 2-Phase-Commit Protokoll verdeutlichen:

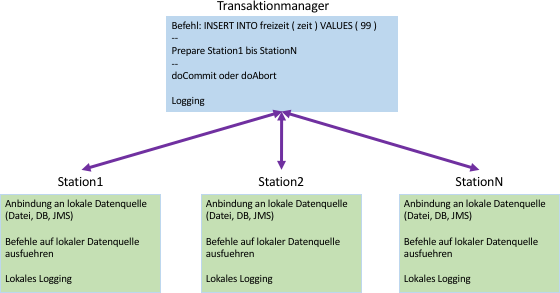


Abbildung 1 Erklärungsgrafik

# Ergebnisse

## Public Key erstellen

Es wird ein PublicKey mit der Asymmetrischen Verschlüsselung verschlüsselt. Dies passiert im Service. Dieser Verschlüsselte PK wird an die Description der Group: group.service1 geschickt und abgespeichert.

private KeyPair generateKeyPair() throws NoSuchAlgorithmException, NoSuchProviderException {  
 KeyPairGenerator keypair = KeyPairGenerator.*getInstance*("RSA");  
 SecureRandom secure = SecureRandom.*getInstance*("SHA1PRNG", "SUN");  
 keypair.initialize(1024, secure);  
 KeyPair keyPair = keypair.generateKeyPair();  
 return keyPair;  
}  
  
public void savePK() {  
 System.*out*.println("Create PK“);  
 this.c.setDescription(toHex(this.keyPair.getPublic().getEncoded()), "service1");  
}

## Symmetrischer Schlüssel

Der Client erzeugt mittels Cipher einen Symmetrischen Schlüssel der mittels des zuvor abgespeicherten Public Keys verschlüsselt wird.

public SecretKey generateSymKey() {  
 try {  
 KeyGenerator keygen = KeyGenerator.*getInstance*("AES");  
 return keygen.generateKey();  
 } catch (NoSuchAlgorithmException e) {  
 e.printStackTrace();  
 }  
  
 return null;  
}  
  
public void sendEncryptSymKey() {  
 try {  
 Cipher cipher = Cipher.*getInstance*("RSA");  
 cipher.init(Cipher.*ENCRYPT\_MODE*, this.pk);  
 byte[] encoded = cipher.doFinal(this.sk.getEncoded());  
 this.client.write(encoded);  
 } catch (NoSuchAlgorithmException e) {  
 e.printStackTrace();  
} catch(NoSuchPaddingException e){  
 e.printStackTrace();  
} catch(BadPaddingException e){  
 e.printStackTrace();  
} catch(InvalidKeyException e){  
 e.printStackTrace();  
} catch(IllegalBlockSizeException e){  
 e.printStackTrace();  
}  
}

## JAVA-Kommunikation

Mittels Sockets die den Host bzw. den Port übernehmen.

Server:

public void start() {  
 try {  
 cSocket = sSocket.accept();  
  
 in = new DataInputStream(cSocket.getInputStream());  
 out = new DataOutputStream(cSocket.getOutputStream());  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
}

Client:

public void start() {  
 try {  
 this.socket = new Socket("192.168.17.128", 55555);  
 this.out = new DataOutputStream(this.socket.getOutputStream());  
 this.in = new DataInputStream(this.socket.getInputStream());  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
}

# Probleme

Derzeit kann nicht mittels Symmetrischer Verschlüsselungsart verschlüsselt werden. Der PublicKey wird zwar vom Naming Service geholt aber der Symmetrische Schlüssel wird nicht mittels des Public Keys verschlüsselt.