

## Vernetzte Systeme, Übungsstunde 10: Hinweise

Gabor Cselle, gabor@student.ethz.ch, <http://www.gaborcselle.com/vs/>, 17. Juni 2005

---

### Organisatorisches

- Vorbesprechung Serie 12, Nachbesprechung Übung 10
- Programmieraufgaben Prüfung: Wattenhofer: nein; Alonso: vielleicht
- Ergebnisse Umfrage Didaktikzentrum. Kommentare:
  - "Was noch möglich wäre: jeweils eine Kurzzusammenfassung des Vorlesungsstoffes der Woche, da es recht schnell vorangeht."
  - "Immer noch zu wenig Kaffee und Kuchen."
  - "Evtl. noch angefügte Ergänzungen updaten (Z.B. Antworten auf die Verständnisfragen in der Übung)"
  - "Es wäre gut, wenn in der Übungsstunde die Serie nachbesprochen würde, die wir an dem Tag abgeben mussten."

### Vorbesprechung Übung 12

#### *Interaktiv*

Wie wartet man bei Java für 1 Sekunde Dauer?

**A.** `s = System.currentTimeMillis(); while(System.currentTimeMillis()- s < 1000) {}`

**B.** `Thread.sleep(1000);`

**C.** `Object.wait(1);`

#### *Lösung*

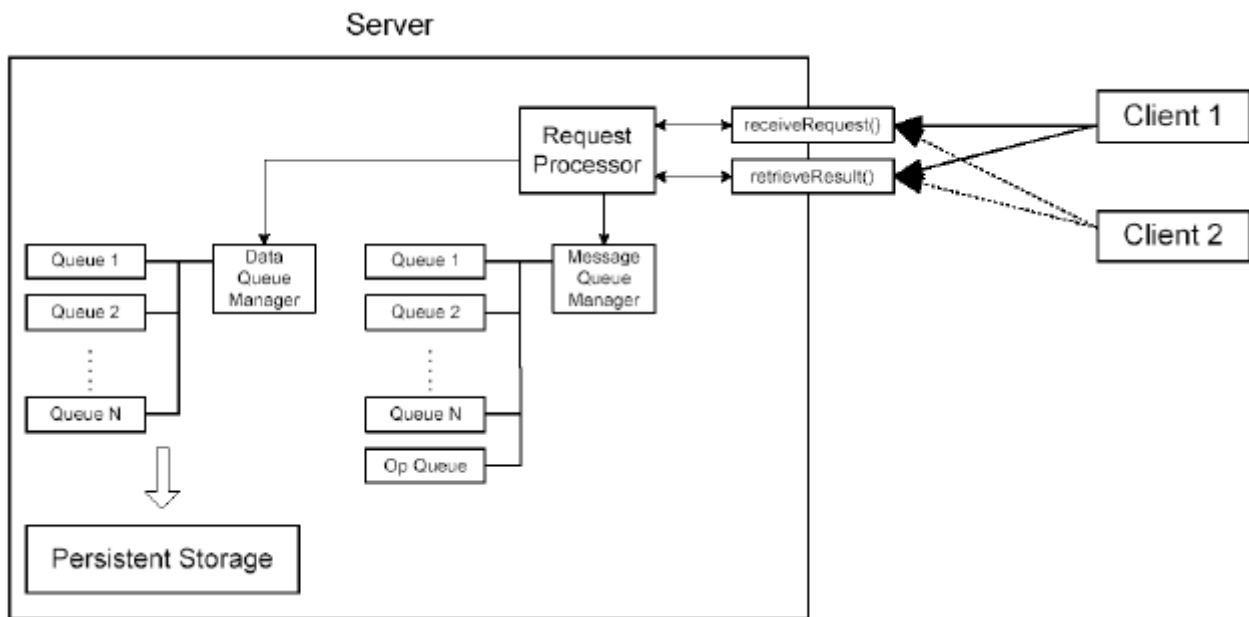
A. Ist "busy waiting", der Prozessor ist voll ausgelastet → bitte vermeiden

C. `Object.wait()` ist ein Aufruf, der für die Threadsynchronisierung gedacht ist

B. Ist die richtige Lösung. Normalerweise schreibt man:

```
try {
    Thread.sleep(1000);
} catch (InterruptedException e) { // evtl. Fehlerbehandlung:
    // [anderer Thread hat uns mit unterbrochen mit ourThread.interrupt();]
}
```

## Aufgabenstellung: Entkopplung von Request und Result.



### Wie führt Client eine Queue-Operation durch (z.B. enqueue / dequeue)?

1. Erstellt RequestMessage-Objekt req, füllt mit Daten der Anfrage
2. Ruft auf RMI-Objekt beim Server `receiveRequest(req)` auf.  
→ Erhält als Rückgabewert ein AckMessage-Objekt ack
3. Ruft auf RMI-Objekt beim Server `retrieveResult(ack)` auf.  
→ Erhält als Rückgabewert ein ResultMessage-Objekt res
  - a. `res.getResType() == RES_TYPE_SUCCEED` → alles prima
  - b. `res.getResType() == EXCEPTION_TYPE_*` → Fehler!

### Was passiert aus Sicht des Servers?

1. Server erhält RequestMessage-Objekt req
2. Nimmt req auseinander führt die darin beschriebene Operation aus. Je nach Art des Requests Ergebnis eintragen:
  - a. enqueue/dequeue request für "queue<X>":  
In ResultMessageQueue "queue<X>"
  - b. create queue / delete queue:  
In ResultMessageQueue "opqueue"

... und etwas später:

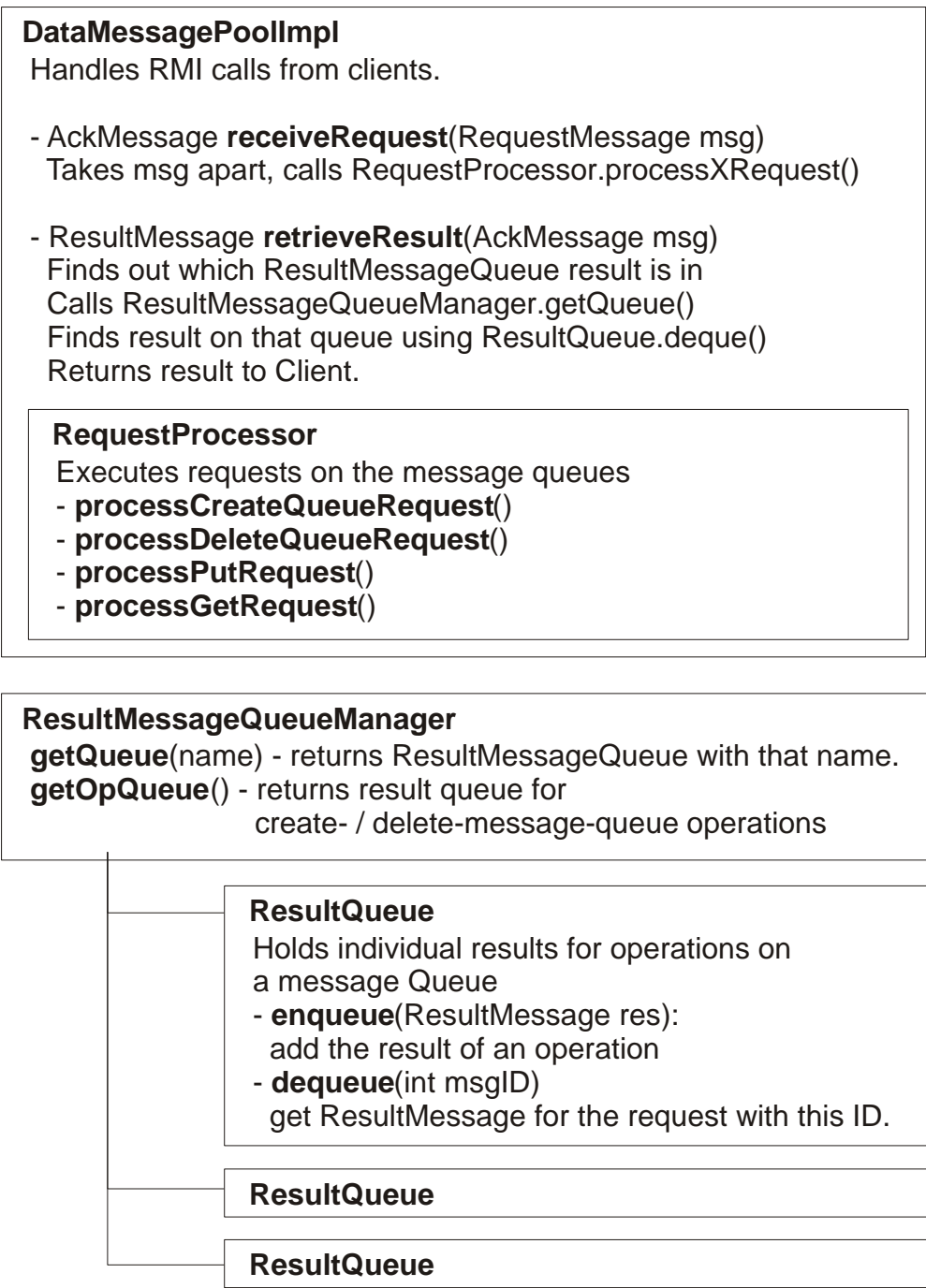
1. Server erhält AckMessage-Objekt ack
2. In `ack.getQueueName()` steht, ob es um "queue<X>" oder "opqueue" geht.
3. In `ack.getMessageNumber()` steht eine eindeutige ID für jeden Aufruf.
4. Mit den Infos aus 2. und 3. wird die richtige Antwortnachricht aus der richtigen ResultMessageQueue geholt und an den Client returned.

### Wofür ist so eine Entkopplung eigentlich gut?

- Batch-Systeme: Sehr lange Bearbeitungszeit. Man platziert Auftrag und holt sich das Resultat irgendwann später ab.

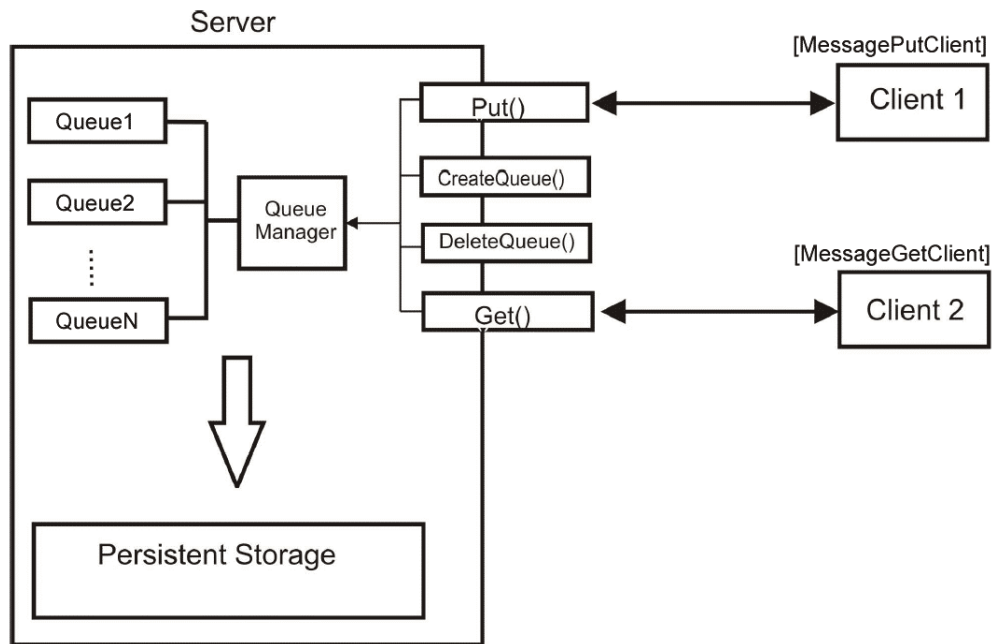
## Architektur des Server-Skeletons

(PersistMessageQueue wie gehabt, hier nicht weiter aufgeführt)



*Frage in der Übungsstunde:* Warum gibt es "normale Queues" (queuex) und "Operations Queues" (opqueue)? *Antwort:* Die Opqueue speichert die Resultate von Create / Delete Queue-Befehlen. Diese müssen wir von den normalen put/get Resultaten in den normalen Queues trennen: Wenn z.B. eine Queue gelöscht wird, wird auch die entsprechende Result Queue gelöscht. Das Resultat für Delete-Befehle können wir also nicht dort speichern. Die Create-Queue-Results können dann praktischerweise auch in die opqueue.

## Nachbesprechung Übung 10



**MessagePutClient** – in main():

```
while (true) {  
    try {  
        String msg = String.valueOf(System.currentTimeMillis());  
        pool.put(msg, arg[0]);  
        System.err.println("Put Msg: " + msg + " into QueueB ");  
        Thread.sleep(wait_time);  
    } //...  
}
```

**MessageGetClient** – in main():

```
while (true) {  
    try {  
        String msg = pool.get(arg[0]);  
        System.out.println("The content of the message" + i + ": " + msg);  
    } //...  
}
```

## PersistMessagePoolImpl

```
public void createQueue(String name) throws RemoteException {
    boolean result = queueManager.createQueue(name);
    if (!result) {
        throw (new QueueDuplicateException("already exists: " + name));
    }
    System.out.println("Output: Create Queue " + name);
}

public void deleteQueue(String name) throws RemoteException {
    boolean result = queueManager.deleteQueue(name);
    if (!result) {
        throw (new QueueNotFoundException("Can't find" + name));
    }
    System.out.println("Output: Delete Queue " + name);
}

public void put(String msg, String queueName) throws RemoteException {
    PersistMessageQueue queue = queueManager.getQueue(queueName);
    if (queue == null)
        throw new QueueNotFoundException("Can't find" + queueName);
    else {
        boolean result = queue.enqueue(msg);
        if (!result) throw new QueueFullException("Q full " + queueName);
    }
    System.out.println("Put msg(" + msg + ") into Q " + queueName);
}

public String get(String queueName) throws RemoteException {
    PersistMessageQueue queue = queueManager.getQueue(queueName);
    if (queue == null)
        throw new QueueNotFoundException("Can't find " + queueName);
    else {
        String msg = queue.dequeue();
        if (msg == null)
            throw new QueueEmptyException("Empty" + queueName);
        System.out.println("Get m(" + msg + ") from Q " + queueName);
        return msg;
    }
}
```

## **PersistMessageQueueManager**

```
private void initPersistStorage() {
    queueDirectory = new File(queueDirectoryName);
    if (!queueDirectory.exists()) {
        queueDirectory.mkdir();
    } else {
        // get a listing of files and add a queue for each (see ML)
    }
}
protected boolean deleteQueue(String name);
    PersistMessageQueue q = (PersistMessageQueue)
        queueList.remove(name);

    if (q == null)
        return false;
    else {
        q.destroyPersistStorage();
        return true;
    }
}
protected void destroyPersistStorage() {
    fileInStream.close();
    fileOutputStream.close();

    queueInStream.close();
    queueOutputStream.close();

    boolean result = queueFile.delete();
}
```

## **PersistMessageQueue**

```
private void updatePersistStorage() {
    fileOutputStream = new FileOutputStream(queueFile, false);
    queueOutputStream = new ObjectOutputStream(fileOutputStream);

    queueOutputStream.writeObject(queue);
    queueOutputStream.flush();

    queueOutputStream.close();
    fileOutputStream.close();
}
public boolean enqueue(String msg) {
    queue.addElement(msg);
    updatePersistStorage();
}
```