

RPC-Semantiken

- Fehler bei Fernafrufen
- Fehlertolerante Fernafrufe
- Übungsaufgabe 3



Fehlerquellen

- Rechner (Client und/oder Server)
 - Prozess-, Programm-, Rechnerabsturz
 - Verzögerungen (z. B. aufgrund von Überlast)
- Kommunikation
 - Nachrichten (Anfrage und/oder Antwort)
 - Reihenfolgeänderung
 - Korrumperung
 - Vervielfachung
 - Verlust
 - Verbindung
 - Verlangsamung
 - Abbruch
- Umgang mit im Fernafruf begründeten Fehlern
 - Fehlerbehandlung im Fernafrufsystem
 - Nur bei Scheitern der Fehlerbehandlung: Signalisierung



Fehlerquellen

- Anwendung
 - Fehler sind nicht im Fernafruf begründet
 - Fehlersituation tritt bei lokalem Methodenaufruf ebenfalls auf
 - Beispiele
 - Falsche Eingaben
[Vergleiche: `IllegalArgumentException` von `VSBoard.get()`]
 - Programmierfehler in der Anwendung
 - ...
- Umgang mit in der Anwendung begründeten Fehlern
 - Aus Sicht des Fernafrufsystems: reguläres Verhalten
 - Keine Fehlerbehandlung im Fernafrufsystem
 - Transparente Signalisierung



Einordnung

- Rechnerfehler
 - Lokaler Methodenaufruf
 - Aufrufer und Aufgerufener in gleichem Maße betroffen
 - Im Fehlerfall sind beide weg bzw. langsam
 - Fernafruf
 - Aufrufer und Aufgerufener können unabhängig ausfallen
 - Im Fehlerfall ist eventuell nur einer betroffen
- Kommunikationsfehler
 - Lokaler Methodenaufruf
 - Keine Netzwerkkommunikation
 - Fehlerart nicht relevant
 - Fernafruf
 - Temporäre oder sogar dauerhafte Fehler möglich
 - Nicht alle Fehler lassen sich im Fernafrufsystem tolerieren
- Konsequenz
 - **Das komplexere Fehlermodell macht es unmöglich Fernafrufe vollständig transparent zu realisieren!**



- Fehlertolerierung
 - Transparente Mechanismen
 - RPC-Semantiken (später mehr)
 - Replikation [Siehe Übungsaufgabe 4.]
 - ...
 - Beliebig komplex
 - Geringer Aufwand → Tolerierung weniger bzw. nur bestimmter Fehlern
 - Hoher Aufwand → Tolerierung vieler Fehlern
 - **Trotzdem: Nicht alle Fehler lassen sich tolerieren**
- Fehlersignalisierung
 - Benachrichtigung an den Benutzer des Fernaufsystems
 - Notwendig wenn Fehler nicht toleriert werden konnten
 - Benutzer des Fernaufsystems muss darauf vorbereitet sein
 - **Verletzung der Transparenzeigenschaften**



Fehlererkennung bei Fernaufufen

- Probleme
 - Keine definitive Fehlererkennung (Liegt überhaupt ein Fehler vor?)
 - Keine exakte Fehlerlokalisierung (Wo liegt der Fehler?)
- Beispiel
 - Szenario: Client erhält keine Antwort auf seine Anfrage
 - Mögliche Gründe
 - Anfrage ging verloren
 - Antwort ging verloren
 - Server ausgefallen
 - Server überlastet
 - Netzwerk überlastet
 - ...
 - Konsequenz: Mindestens einer der beiden Fernauf-Teilnehmer kann nicht erkennen, ob (und wenn ja wo) ein Fehler vorliegt
- Eine präzise Fehlererkennung ist in verteilten Systemen im Allgemeinen nicht möglich!



- Exception-Klasse `java.rmi.RemoteException`
 - Oberklasse für alle Fernauf-Ausnahmesituationen
 - Muss von jeder Methode einer Remote-Schnittstelle geworfen werden
→ **Verletzung der Zugriffstransparenz**
 - Beispiel
- Unterklassen von `RemoteException`
 - `ConnectException`: Verbindungsauflauf fehlgeschlagen
 - `NoSuchObjectException`: Remote-Objekt nicht (mehr) verfügbar
 - `ServerError`: Auspacken der Anfrage, Ausführung der Methode oder Einpacken der Antwort fehlgeschlagen
 - `UnknownHostException`: Remote-Host nicht bekannt
 - ...



Überblick

RPC-Semantiken

- Fehler bei Fernaufufen
- Fehlertolerante Fernaufufe
- Übungsaufgabe 3



Was beinhaltet Fehlertoleranz für Fernauftrufe?

- Probleme
 - Unpräzise Fehlererkennung kann zu inkonsistenten Sichtweisen führen, z. B. bei einer verlorenen Antwortnachricht:
 - Client geht von einem Fehler aus, da er keine Antwort erhalten hat
 - Server befindet sich im Normalzustand, da er die Antwort gesendet hat
 - Wiederherstellung einer konsistenten Sichtweise erfordert weitere Kommunikation zwischen Client und Server
 - Diese Nachrichten können ebenfalls Fehlern unterliegen
 - Erneutes Senden und Ausführen einer Anfrage kann zu unerwünschten Zuständen führen
- Allgemeine Hilfsmittel
 - Duplikaterkennung
 - Idempotente Operationen
 - Rollbacks (→ Transaktionen)
 - ...



Fernauftrufsemantiken

- Bemerkung
 - Im Folgenden wird die Tolerierung von **Kommunikationsfehlern** betrachtet, Rechnerausfälle werden ausgeklammert
 - Die Tolerierung von Rechnerauffällen erfordert Mechanismen zum Wiederanlaufen (z. B. Zustandswiederherstellung)
- Semantiken
 - Maybe
 - At-Least-Once
 - At-Most-Once
 - Last-Of-Many
- Unterschiede
 - Mehrmaliges Senden von Anfragen
 - Aktualität der Antworten
 - Anzahl der Ausführungen → *Idempotente Operationen?*
 - Antwortspeicherung → Wie lange wird eine Antwort aufgehoben?



Kombinierter Ansatz

- Fehlertolerantes Kommunikationsprotokoll
 - Beispiel: TCP statt UDP
 - Tolerierung von
 - Reihenfolgeänderung
 - Korrumperung
 - Vervielfachung
 - Verlust
 - von Nachrichten
- Aufrufsemantiken
 - Wiederherstellung konsistenter Sichtweisen von Client und Server
 - Tolerierung von Verzögerungen (Rechner und/oder Netzwerk)
- Hinweise
 - Alle durch das Kommunikationsprotokoll tolerierte Fehler sind ersatzweise durch eine entsprechende Aufrufsemantik tolerierbar
 - Die optimale Kombination von Kommunikationsprotokoll und Aufrufsemantik hängt vom Einzelfall ab



Idempotenz

- Idempotente Funktionen (Mathematik)
 - Definition
 - $f(x) = f(f(x))$
 - Beispiele
 - $f(x) = c$ (konstante Funktion)
 - $f(x) = x \cdot 1$ (Multiplikation mit 1)
 - $f(x) = \frac{x}{1}$ (Division durch 1)
- Idempotente Operationen (Informatik)
 - Eigenschaften
 - Mehrfache Ausführung erzeugt stets den selben Rückgabewert
 - Mehrfache Ausführung hinterlässt die Anwendung auf dem Server in stets dem selben Zustand
 - Bankautomat-Beispiel
 - Lösungsansatz: Verwendung absoluter Beträge
 - Achtung: nicht-triviale Implementierung, da zusätzliche Synchronisation und/oder Ausnahmebehandlung nötig



Antwortspeicherung

- Problem
 - Server stellt eigene Ressourcen (→ Antwort-Cache) für fehlertolerante Feraufrufe von Clients bereit
 - Mit jedem neuen Feraufruf werden zusätzliche Ressourcen belegt
 - Wann können diese Ressourcen freigegeben, d. h. gespeicherte Antworten verworfen werden?
- Lösungsansätze (Kombinationen möglich bzw. nötig)
 - Explizit
 - Benachrichtigung durch Client oder Nachfrage vom Server
 - Problem: Nicht alle Clients können oder wollen sich daran halten
 - Implizit
 - Bei neuem Feraufruf eines Client wird die alte Antwort gelöscht
 - Problem: Letzter Feraufruf eines Client
 - Timeout
 - Antwortlöschung nach Ablauf eines feraufrufspezifischen Timeout
 - Als Rückfallposition immer nötig



Last-Of-Many

- Funktionsweise
 - Client wiederholt Anfrage, falls Antwort ausbleibt
 - Client akzeptiert nur Antwort auf seine aktuellste Anfrage
- Implementierung
 - Feraufruf muss eindeutig identifizierbar sein
 - Client
 - Remote-Objekt
 - Remote-Methode
 - Aufrufzähler
 - Jede Feraufrufnachricht muss eindeutig identifizierbar sein
 - Anfragezähler
 - Zuordnung: Antwort zu Anfrage
- Eigenschaften
 - Keine Antwortspeicherung nötig
 - Anfragen werden eventuell mehrfach ausgeführt



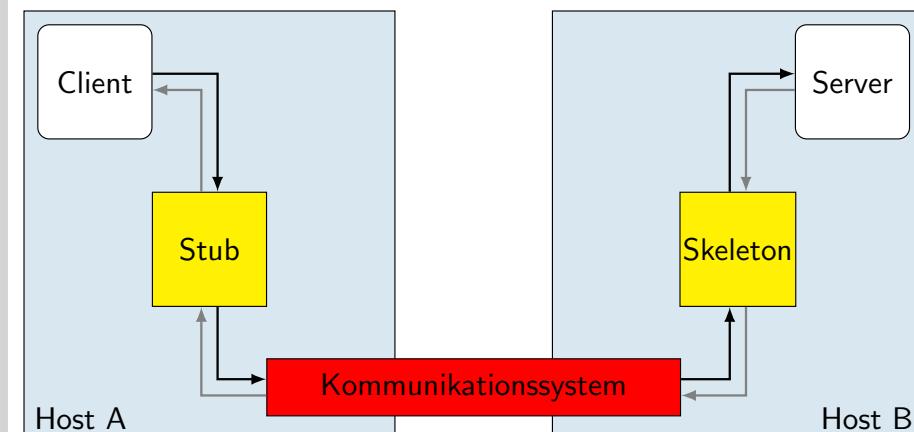
At-{Least,Most}-Once

- At-Least-Once (z. B. SUN RPC)
 - Funktionsweise
 - Client wiederholt Anfrage, falls Antwort ausbleibt
 - Client akzeptiert die erste Antwort, die ihn erreicht
 - Eigenschaften
 - Anfragen werden eventuell mehrfach ausgeführt
 - Client verwendet eventuell veraltete Antwort
- At-Most-Once (z. B. Java RMI)
 - Funktionsweise
 - Client wiederholt Anfrage, falls Antwort ausbleibt
 - Server speichert Antwort
 - Server sendet bei Anfragewiederholungen gespeicherte Antwort
 - Eigenschaften
 - Anfragen werden höchstens einmal ausgeführt
 - Speichern von Antworten erforderlich



Übungsaufgabe 3

- Simulation von Kommunikationsfehlern
- Bereitstellung von Fehlertoleranzmechanismen



- Simulation von Kommunikationsfehlern
 - Verlust von Nachrichten
 - Verzögerung einzelner Nachrichten
 - Vervielfachung von Nachrichten
 - Nicht betrachtet: Fehlerhafte Teilnachrichten
- Tests
 - Variation der Fehlerintensität
 - Kombination verschiedener Fehlerarten
- Mögliche Implementierung
 - Fehlerhafte VSObjectConnection → VSObjectConnection
 - Überschreiben von
 - sendObject() oder
 - receiveObject()



Mögliche Timeout-Behandlung in Java

- Timer-Klasse java.util.Timer
 - Einfache Scheduler-Funktionalität
 - Unterstützung ein- sowie mehrmaliger Task-Ausführung
 - Task-Objekte: Instanzen von TimerTask-Unterklassen
 - Wichtige Methoden

```
void schedule(TimerTask task, long delay);
void scheduleAtFixedRate(TimerTask t, long dy, long period);
void cancel();
- schedule()          Einmalig auszuführenden Task aufsetzen
- scheduleAtFixedRate Periodischen Task aufsetzen
- cancel()            Timer beenden
```
- Timeout-Handler-Klasse java.util.TimerTask
 - Timeout-Handler
 - Basisklasse für von Timer eingeplante Tasks
 - Auszuführenden Task-Code in Unterklassen implementieren
 - Wichtige Methoden

```
abstract void run();
boolean cancel();
- run()              Task ausführen → Timeout behandeln
- cancel()           Task bzw. Timeout abbrechen
```



- *Last-Of-Many*
 - Fernaufruf-IDs
 - Sequenznummern
 - Timeouts
- *At-Most-Once*
 - Einmalige Ausführung
 - Speicherung der Ergebnisse
 - Garbage Collection für Ergebnisse
- Hinweis
 - Die zum Einsatz kommende Fernaufrufsemantik
 - *Maybe*
 - *Last-Of-Many*
 - *At-Most-Once*
 - soll konfigurierbar sein!



Timer/TimerTask-Beispiel

```
public class VSTimerExample {
    public static void main(String[] args) {
        Timer timer = new Timer();
        TimerTask handler = new VSTimeoutHandler();

        // Timeout auf 5 Sekunden setzen
        timer.schedule(handler, 5000);

        // Zu ueberwachenden Code ausfuehren
        [...]

        // Timeout deaktivieren und Timer aufraeumen
        handler.cancel();
        timer.cancel();
    }
}

class VSTimeoutHandler extends TimerTask {
    public void run() {
        System.err.println("ALARM!");
    }
}
```

