

# Verteilte Systeme – Übung

## Rückrufe bei Fernaufrufen

---

Sommersemester 2022

Laura Lawniczak, Tobias Distler

Friedrich-Alexander-Universität Erlangen-Nürnberg  
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)

[www4.cs.fau.de](http://www4.cs.fau.de)



Lehrstuhl für Verteilte Systeme  
und Betriebssysteme



FRIEDRICH-ALEXANDER  
UNIVERSITÄT  
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Rückrufe

## Rückrufe

---

## ■ Beispielszenario [Vergleiche Übungsaufgabe 1.]

### ▪ Server-Seite

```
public interface VSAuctionService {  
    public void registerAuction(VSAuction auction, int duration, VSAuctionEventHandler handler)  
        throws VSAuctionException;  
    public VSAuction[] getAuctions();  
    public boolean placeBid(String userName, String auctionName,  
        int price, VSAuctionEventHandler handler)  
        throws VSAuctionException;  
}
```

### ▪ Client-Seite

```
public interface VSAuctionEventHandler {  
    public void handleEvent(VSAuctionEventType event, VSAuction auction);  
}
```

→ Der Server muss den Client (per Fernaufruf) zurückrufen können

→ Dem Server muss eine Referenz auf den Client vorliegen

## ■ Lokaler Methodenaufruf

- Identischer Adressraum
  - Referenz auch in aufgerufener Methode gültig
- Rückruf erfordert keine spezielle Betrachtung

## ■ Fernaufruf

- Unterschiedliche Adressräume
  - Referenz normalerweise nicht in aufgerufener Methode gültig  
[Ausnahme: z. B. „Distributed Shared Memory (DSM)“-Systeme]
- Einfache Übertragung einer Referenz (meist) nicht sinnvoll

→ Spezielle Semantiken für Parameterübergabe bei Fernaufrufen notwendig

## „Rückruf“ per **Call-by-Value-Result**

### ■ Funktionsweise

- Dem Server wird eine Kopie des Originalobjekts übergeben
- Aufgerufene Methode kann Kopie modifizieren
- Kopie wird an Client zurückgesendet
- Originalobjekt wird durch Kopie ersetzt

### ■ Vorteile

- Einfache Implementierung (→ Serialisierung)
- Ermöglicht direkte Speicherzugriffe

### ■ Nachteile

- Gültigkeit der Referenz ist beschränkt auf Methodenausführung
- Komplettes Objekt wird doppelt übertragen
- Verkompliziert Synchronisation, Zugriff auf Ressourcen

## Rückruf per **Call-by-Reference**

- Funktionsweise
  - Objekt wird auf Client-Seite für Fernaufrufe verfügbar gemacht
  - Dem Server wird als Parameter eine **Remote-Referenz** übergeben
  - Jeder Server-seitige Zugriff auf das Objekt erfolgt per Fernaufruf
  - Aufgerufene Prozedur kann Daten des Aufrufers direkt verändern
- Vorteile gegenüber Call-by-Value-Result
  - Speicherung der Referenz für spätere Verwendung möglich
  - Geringere zu übertragende Datenmenge bei großen Objekten mit wenigen Zugriffen
- Nachteil
  - Benötigt spezielle Unterstützung für Speicherzugriffe

## Call-by-Reference in objektorientierten Programmiersprachen

### ■ Funktionsweise

- Objekt kapselt Daten
- Idealfall: Zugriff nur über Methodenaufrufe
- Übertragung einer Remote-Referenz führt auf Server-Seite zur Erzeugung eines Objekt-Stub
- Server kann transparent auf das Originalobjekt zugreifen

### ■ Einschränkung

- Kein direkter Zugriff auf Objektzustand
- z. B. keine „public“-Variablen

→ Problem ohne spezielle Unterstützung durch Betriebssystem bzw. Laufzeitumgebung lösbar



- Naiver Ansatz
  - Bei jeder Weitergabe einer Objektreferenz werden ein neuer Stub sowie ein neuer Skeleton erzeugt  
→ Unnötig, falls dieselbe Objektreferenz mehrfach übertragen wird
- Mögliches Verfahren in Fernaufrufsystemen: Beidseitiger Einsatz von Hash-Tabellen
  - Client-Seite: Zuordnung lokaler Objektreferenzen auf Remote-Referenzen
  - Server-Seite: Abbildung von Remote-Referenzen auf Stubs

Wie lange sollen diese Informationen verfügbar gehalten werden?

## ■ Allgemein

### ■ Wo?

- Auf Applikationsebene
- Bei Rückrufen: Im Skeleton des Originalfernaufrufs (auf Server-Seite)

### ■ Wie?

- Explizit: z. B. konkrete Anweisung
- Implizit: z. B. Methodenende
- Automatisiert: z. B. Garbage-Collection

## ■ Java RMI

- Reguläre Garbage Collection: Stub wird gelöscht, sobald keine Referenz mehr auf ihn verweist
- Zusätzlich: Distributed Garbage Collection für Remote-Referenzen

- Jeder Server unterhält je einen Remote-Referenzen-Zähler auf von ihm bereitgestellte Remote-Objekte
  - `dirty()`-Methode
    - Inkrementiert den Zähler
    - Aufgerufen vom Client bei Stub-Erzeugung (per Fernaufruf)
  - `clean()`-Methode
    - Dekrementiert den Zähler
    - Aufgerufen vom Client bei Stub-Freigabe (per Fernaufruf)
  
- Lokal bereitgestelltes Remote-Objekt wird vom Server der Garbage Collection überlassen, sobald
  - keine lokalen Referenzen mehr auf das Objekt existieren **und**
  - der Remote-Referenzen-Zähler auf Null steht

- **Leases** im Kontext von Fernaufrufen:  
Garantie des Servers an den Client, dass ein bestimmtes Remote-Objekt für eine gewisse Zeit verfügbar ist
- Leases in Java RMI
  - Standarddauer pro Lease: 10 Minuten
  - Rückgabewert von `dirty()`-Aufrufen
  - Verlängerung durch erneuten Aufruf von `dirty()`  
[Erfolgt üblicherweise nach Ablauf der Hälfte der Lease-Dauer.]
  - Ablauf eines Lease
    - Dekrementieren des entsprechenden Remote-Referenzen-Zählers
    - Bei Bedarf: Garbage-Collection des Stubs

→ Leases sind eine Absicherung des Servers gegen Verbindungsausfälle und Client-Abstürze