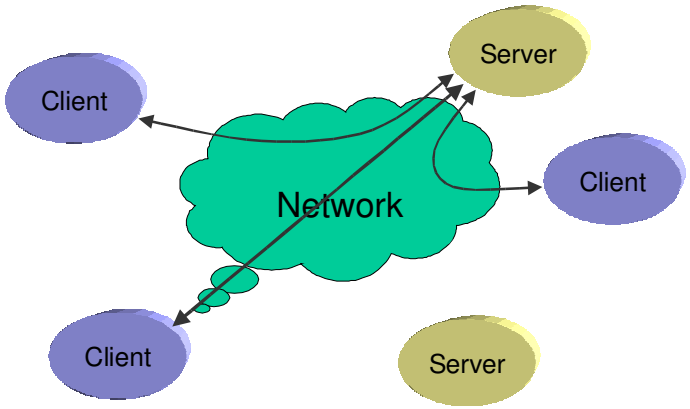
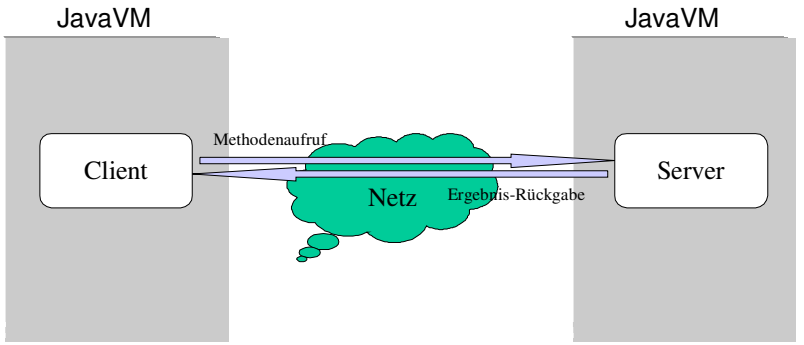
Remote Method Invocation (RMI)



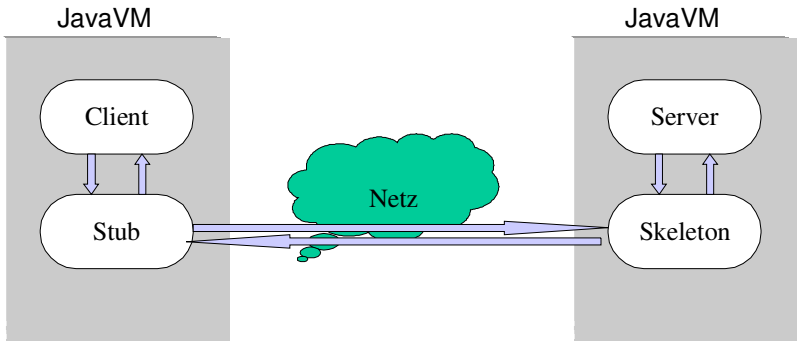
RMI



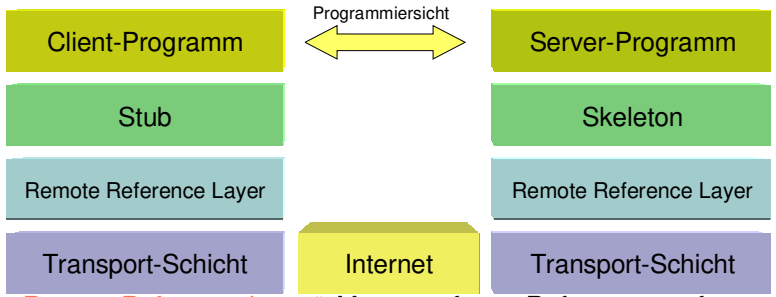
RMI

- RMI = „entfernter Methodenaufruf“
- „Klassisches“ Java: Methodenaufrufe innerhalb einer einzelnen Java Virtuellen Maschine (JVM)
- Mit RMI können Objekte in einer JVM Objekte in einer entfernten JVM aufrufen
- JVMs können auf unterschiedlichen Rechnern im Netz ausgeführt werden
- Bekannte Idee: Erzeuge Stellvertreter des entfernten Objektes in lokaler JVM
  - Client-Stub und Server-Skeleton (übersetzt mit ***`rmic`***)

RMI: Stubs und Skeleton



RMI-Schichtenmodell



- „Remote Reference Layer“: Mappt entfernte Referenzen auf Rechnernamen und Objekte; verwaltet darunterliegende TCP-Verbindung

RMI-Programmierung

- Entfernte Methoden werden durch „remote interfaces“ definiert:

```
import java.rmi.*;

public interface Hello extends Remote {
   public String sayHello() throws RemoteException;
}
```

Methoden können immer `RemoteException` auslösen

- Das entfernte Objekt kann nicht mehr verfügbar sein
- Das Netz kann gestört sein

RMI-Programmierung

- Server-Programme impl. Interface und erweitern i.a. `java.rmi.UnicastRemoteObject`
  - Implementierung der entfernten Klassen
  - `UnicastRemoteObject` stellt die wichtigsten Methoden für die Verwendung von RMI bereit

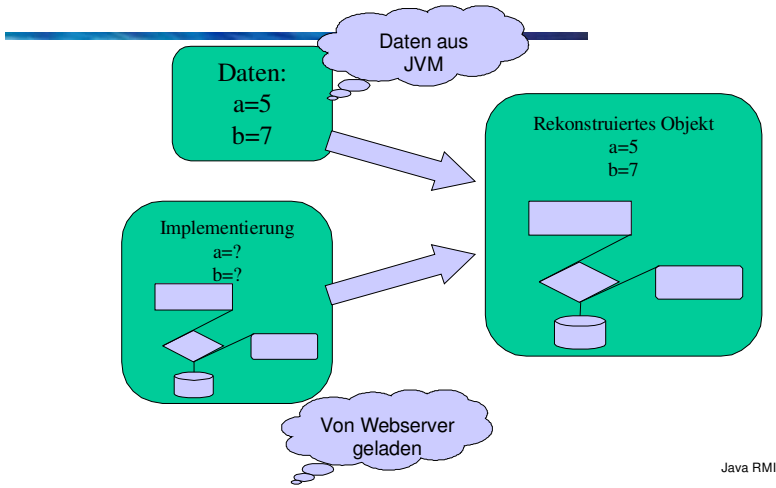
```
import java.rmi.*;
import java.rmi.server.*;
import java.net.*;

public class HelloImpl implements Hello extends UnicastRemoteObject {
   public HelloImpl() throws RemoteException {
}
   public String sayHello() throws java.rmi.RemoteException {
     return „hallo“;
   }
}
```

Vollständiges „Hello World“ Beispiel: http://www.ccs.neu.edu/home/kenb/com3337/rmi_tut.html

Code-Mobilität

- RMI überträgt Zustand eines Objekts, nicht aber Implementierung der zugehörigen Klasse
- Problem: Klassen sind i.a. beim Client nicht lokal verfügbar (im Klassenpfad aufgelistet)
  - Klassen-Implementierungen müssen zur Laufzeit des Client dynamisch aus dem Netz nachgeladen werden
  - Auf Empfängerseite werden Zustand und Implementierung zu gültigem Objekt zusammengefügt
- Sicherheit?

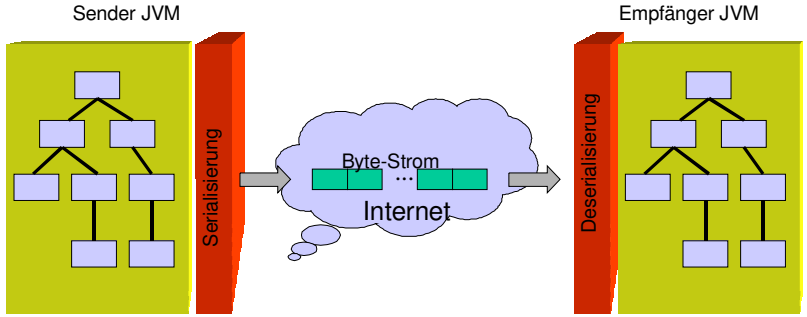
Code-Mobilität



Serialisierung

- Methoden-Parameter können auf zwei Arten verschickt werden:
  - „Remote object“ wird als „remote reference“ transportiert („verlässt“ JVM nicht)
  - Als Kopie des Objektes („call by value“)
- Kopien werden zu fremder JVM übertragen
  - Datenstrukturen müssen in Datenstrom umgewandelt werden
  - Transportierter Datenstrom muss beim Empfänger in Datenstruktur verwandelt werden
  - Serialisierung-API in Java

Serialisierung



Serialisierung

- Transformation eines Objektes in einen Bytestrom
  - Objekt besteht aus seinem Zustand und der Implementierung in Form einer `class`-Datei
  - Zustand ist dynamisch und nur zur Laufzeit bekannt
  - Übertrage einen „Schnappschuss“ dieses Zustands
- Problem: Referenzen auf andere Objekte
  - Nicht nur Referenzen auf primitive Typen
  - Alle referenzierten Objekte müssen ebenfalls rekursiv serialisiert werden
  - Bilde transitive Hülle über alle referenzierten Objekte
  - Endlichkeit!

Serialisierung - Verwendungs-möglichkeiten

- Zustand kann serialisiert in Datei gespeichert werden
  - Konfigurationen
  - Spätere Wiederaufnahme einer Bearbeitung eines Zustands
- Zustand kann serialisiert auf einen „Socket“ geschrieben werden
  - Übertragung des Zustands von einer JVM zu einer anderen

Bootstrap

- Weitere serializable/remote object
  - Übergabe als Parameter beim Aufruf eines remote object
  - Erhalt als Rückgabewert eines remote object
- Wie aber erhalte ich das erste remote object?

RMI Registry

- Registrieren von remote objects (bind, rebind, unbind)
- Finden von remote objects (lookup, list)
- Schlüssel sind Namen (strings)
- Bind, rebind, unbind nur von lokalen Clients

```
package java.rmi.registry;

public interface Registry extends java.rmi.Remote {public static final int REGISTRY_PORT =
1099;

public java.rmi.Remote lookup(String name) throws java.rmi.RemoteException,
java.rmi.NotBoundException, java.rmi.AccessException;

public void bind(String name, java.rmi.Remote obj) throws java.rmi.RemoteException,
java.rmi.AlreadyBoundException, java.rmi.AccessException;

public void rebind(String name, java.rmi.Remote obj) throws java.rmi.RemoteException,
java.rmi.AccessException;

public void unbind(String name) throws java.rmi.RemoteException, java.rmi.NotBoundException,
java.rmi.AccessException;

public String[] list() throws java.rmi.RemoteException, java.rmi.AccessException; }
```