Verteilte Systeme - 4. Übung

Tobias Distler, Michael Gernoth, Reinhard Tartler

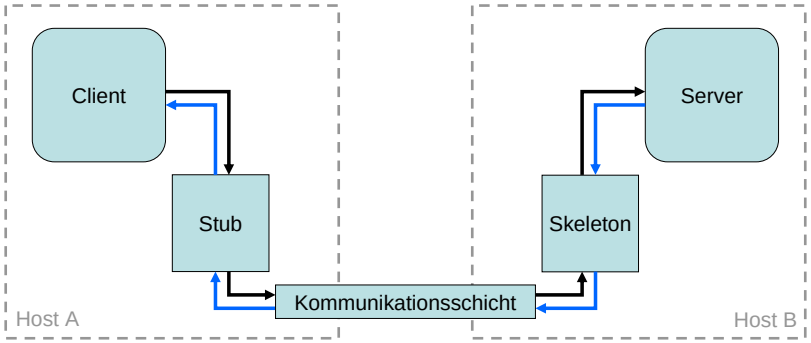
Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
www4.informatik.uni-erlangen.de

Sommersemester 2008

Überblick

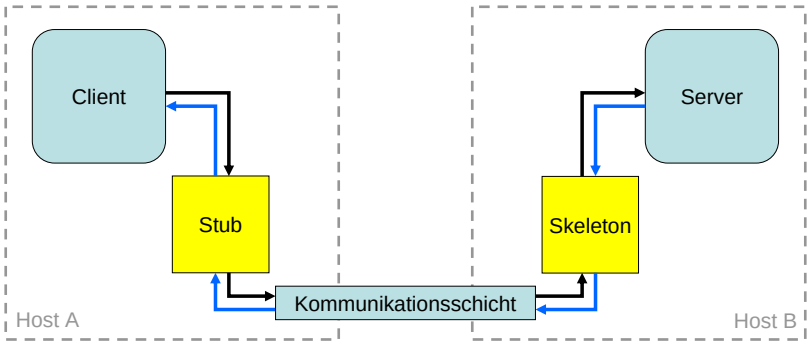
VS-Übung

- Entwicklung eines eigenen Fernaufrufsystems
- Orientierung an Java RMI



Übungsaufgabe 3

- Entwicklung eines Kommunikationssystems
- Dynamische Erzeugung von Stub und Skeleton



Stubs & Skeletons

Grundprinzip des Fernaufrufs

- Umsetzung von Methodenaufruf in Nachrichtenaustausch
- Dabei Abstraktion der Örtlichkeit von Auftraggeber und -nehmer
- Erfordert Ver-/Entpacken von Parametern/Rückgabewerten in Nachrichten

Methodenstümpfe

- Client-Stub und Server-Stub (bzw. -Skeleton) kapseln obige Aufgaben
- Ziel: Automatische Erzeugung der Stümpfe

Verschiedene Lösungen in realen Systemen

- SUN RPC
  - Explizite Schnittstellen-Beschreibungssprache
  - Eingabeparameter ,,by value", Rückgabewert ,,by value"
- Java RMI
  - Auslegung von Parameter direkt in der Sprache beschreibbar
  - Eingabeparameter ,,by value", falls Basisklasse ,,Serializable"; Eingabeparameter ,,by (object) referenze", falls Basiskl. ,,RemoteObject"
- CORBA
  - Explizite Schnittstellen-Beschreibungssprache; Explizite Spezifikation von ,,by value" (in), ,,by result" (out) oder ,,by value/result" (in/out)
  - Parameter vom Typ ,,interface" als entfernte Referenzen
- Microsoft .NET
  - Schnittstelle in Bytecode beschrieben; Auslegung der Parameter durch Basiskl. (MarshalByRefObject / MarshalByValueObject)

Aufgabe des Stub

- Bereitstellung einer Implementierung der Interfaces des entfernten Objekts
- Übergabe von Objekt-ID, Methodenname und Aufrufparameter an das Kommunikationssystem
- Empfang des Rückgabewerts vom Kommunikationssystem
- Übergabe des Rückgabewerts an den Aufrufer

Manuelle Interfaceimplementierung

```
class HelloStub implements HelloInterface {
    public void setName(String name) {
        //ObjID, Methodenname, Parameter an Komm.sys uebergeben
    }
    public String getName() {
        //ObjID, Methodenname an Komm.sys uebergeben
        //Rueckgabewert entgegennehmen
        return ret;
    }
    public void sayHello() {
        //ObjID, Methodenname an Komm.sys uebergeben
    }
}
```

- Hoher Aufwand bei initialer Implementierung und späteren Änderungen am Interface
- Codeduplikation
- Fehleranfällig

Dynamische Proxies

- Zur Laufzeit generierte Stellvertreterobjekte
- Implementieren gewünschte Interfaces
- Rufen für jeden Methodenaufruf am Stellvertreterobjekt einen angegebenen InvocationHandler auf
- Eigene Implementierung des InvocationHandler nötig
- Weiterführende Dokumentation:

`http://java.sun.com/j2se/1.3/docs/guide/reflection/proxy.html`

`http://www.roseindia.net/javatutorials/dynamic_proxies_tutorial.shtml`

Funktionsweise

- Erzeugung eines Objekts zur Laufzeit, welches alle Methoden der angegebenen Interfaces implementiert
- Abbildung aller Methodenaufrufe auf die `invoke`-Methode eines InvocationHandler-Objekts
- Übergabe des Proxy-Objekts, der ursprünglichen Methode und der Methodenparameter im `invoke`-Aufruf
- Rückgabeobjekt der `invoke`-Methode wird zum Rückgabewert des ursprünglichen Methodenaufrufs
- Das erzeugte Proxy-Objekt kann auf alle von ihm implementierten Interfaces gecastet werden
- Typsicherheit ist gewährleistet
- Dynamische Proxies sind Teil der Reflection-API

Invocation Handler

Jeder Methodenaufruf an einer Proxy-Methode wird auf die `invoke`-Methode eines Objekts abgebildet, welches das Interface java.lang.reflect.InvocationHandler implementiert:

```
public Object invoke(Object proxy, Method method,
                     Object[] args) throws Throwable
```

- `proxy`: Das dynamische Proxy-Objekt, welches die `invoke`-Methode aufgerufen hat.
- `method`: Das java.lang.reflect.Method-Objekt der aufgerufenen Proxy-Methode.
- `args`: Object-Array mit den Parametern des ursprünglichen Methodenaufrufs. Falls kein Parameter übergeben wurde ist `args == null`, ansonsten kann die Anzahl über `args.length` bestimmt werden.

Erzeugung

Ein dynamischer Proxy wird durch den Aufruf der Methode java.lang.reflect.Proxy.newProxyInstance erzeugt:

```
static Object newProxyInstance(ClassLoader loader,
                               Class[] interfaces, InvocationHandler h)
```

- `loader`: ClassLoader der Interfaces. Kann bestimmt werden durch den Aufruf von *`Interface`*`.class.getClassLoader()`
- `interfaces`: Array der durch den Proxy zu implementierenden Interface-Klassen (*`Interface`*`.class`)
- `InvocationHandler`: Instanz einer Klasse, welche das InvocationHandler-Interface implementiert

Nach der Erzeugung des Proxy-Objekts kann dieses als Stellvertreter für eine echte Implementierung der angegebenen Inetrfaces genutzt werden.

Beispiel - Interface und Implementierung

```
public interface HelloInterface {
    public void setName(String name);
    public String getName();
    public void sayHello();
}
```

```
public class HelloImpl implements HelloInterface {
    private String name;

    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void sayHello() {
        System.out.println("Hallo " + name);
    }
}
```


Beispiel - InvocationHandler

```
import java.lang.reflect.*;

public class HelloInvocationHandler
  implements InvocationHandler {
    private HelloInterface obj;

    HelloInvocationHandler(HelloInterface obj) {
        this.obj = obj;
    }

    public Object invoke(Object proxy, Method method,
      Object[] args) throws Throwable {
        System.out.println("Methode: " +
          method.toGenericString());
        if (args != null) {
            System.out.println(args.length + " Argument(e)");
        }

        return method.invoke(obj, args);
    }
}
```


Beispiel - Proxyerzeugung und -benutzung

```
import java.lang.reflect.Proxy;
public class HelloTest {
    public static void main(String[] args) {
        HelloInterface hello = new HelloImpl();

        ClassLoader loader =
          HelloInterface.class.getClassLoader();
        Class[] interfaces =
          new Class[] { HelloInterface.class };
        HelloInvocationHandler handler =
          new HelloInvocationHandler(hello);

        HelloInterface helloProxy =
          (HelloInterface)Proxy.newProxyInstance(
            loader, interfaces, handler);

        helloProxy.setName("Benutzer");
        helloProxy.sayHello();
        System.out.println(helloProxy.getName());
    }
}
```


Beispiel - Ausführung

```
faui05 [~/VSHello]> java HelloTest
Methode: public abstract void
         HelloInterface.setName(java.lang.String)
1 Argument(e)
Methode: public abstract void
         HelloInterface.sayHello()
Hallo Benutzer
Methode: public abstract
         java.lang.String HelloInterface.getName()
Benutzer
```

- Jeder Aufruf einer Methode an dem Objekt helloProxy wird durch den dynamisch generierten Proxy an die Methode HelloInvocationHandler.invoke(...) weitergegeben.
- Im verteilten Fall soll im Handler der Fernaufruf des entfernten Objekts erfolgen.

Aufgabe des Skelton

- Empfang von Objekt-ID, Methodenname und Aufrufparameter vom Kommunikationssystem
- Suchen des angegebenen Objekts
- Suchen der übergebenen Methode
- Aufrufen der Methode
- Übergabe des Rückgabewerts an das Kommunikationssystem

Auswahl der richtigen Methode

Methoden werden von der Gegenseite durch einen generischen String (*method*`.toGenericString()`) spezifiziert:

```
public abstract void HelloInterface.setName(java.lang.String)
```

- Sichtbarkeit
- Rückgabewert
- Name der Methode mit Interface
- Parameter

Aufruf der Methode an dem Remote-Objekt auf der Serverseite:

- Abfrage aller Interfaces des Remote-Objekts
- Abfrage aller Methoden der Interfaces
- Vergleich der generischen Strings der Methoden mit übergebenem String
- Aufruf der gefundenen Methode, falls vorhanden

Nachtrag zu Übungsaufgabe 2

„Mit `final` gekennzeichnete Felder müssen nicht übertragen werden"

Unterscheidung notwendig:

- `static final`
  - Müssen bei ihrer Deklaration initialisiert werden
  - Klassenweit eindeutig (→ beim Empfänger bekannt)

  → Nicht übertragen!
- `final` (nicht `static`)
  - Können auch erst im Konstruktor initialisiert werden:

```
public class FinalExample {
    public final int MAX_VALUE;

    public FinalExample(int maxValue) {
        MAX_VALUE = maxValue;
    }
}
```

  - Je nach Objekt evtl. unterschiedlich

  → Übertragen!

Nachtrag zu Übungsaufgabe 2

Objekt senden & empfangen

```
[...]
// {S,Des}erialisierung aller Attribute eines Objekts
for(Field field: objectFields) {
    int mods = field.getModifiers();

    // Feld ignorieren, falls es 'static final' ist
    // Feld uebertragen, falls es NUR 'final' ist
    if(Modifier.isStatic(mods) && Modifier.isFinal(mods)) {
        continue;
    }

    // Feld ignorieren, falls es 'transient' ist
    if(Modifier.isTransient(mods)) continue;

    [...]
}
[...]
```

→ Damit lassen sich jetzt auch Java-Strings übertragen

Kommunikationssystem - Adressierung

- IP-Adresse:
  - DNS-Form: www4.informatik.uni-erlangen.de
  - durch Punkte getrenntes Quadtupel: 131.188.34.200
- `java.net.InetSocketAddress` enthält IP-Adresse und Port
- `InetSocketAddress` kann mit verschiedenen Konstruktoren erzeugt werden:
  - `InetSocketAddress(Strings Hostname, int port)`: Erzeugung mit Hostname und Protnummer
  - `InetSocketAddress(InetAddress addr, int port)`: Erzeugung mit InetAddress-Objekt und Portnummer
  - `InetSocketAddress(int port)`: Erzeugung nur mit Portnummer, als Adresse wird `0.0.0.0` gesetzt

```
InetSocketAddress inetsockaddr = new InetSocketAddress
    ("www4.informatik.uni-erlangen.de", 80);
```


Kommunikationssystem - TCP Client-Sockets

- `java.net.Socket`
  - TCP/IP
  - zuverlässig
  - Repräsentiert einen Kommunikationsendpunkt bei einem Client oder einem Server
- Erzeugen eines neuen Sockets:

```
Socket socket = new Socket();
```

- Verbindung herstellen:

```
socket.connect(inetsockaddr);
```

- Ein Kommunikationsendpunkt ist definiert durch Rechnername und Port (Ports: 16 bit, <1024 privilegiert)
- `close` schließt den Socket.

Kommunikationssystem - TCP Server-Sockets

- `java.net.ServerSocket`
  - wird serverseitig verwendet um auf Verbindungsanfragen von Clients zu warten
- `accept` wartet auf Verbindungsanfragen
- für eine neue Verbindung wird ein neues Socket-Objekt zurückgegeben:

```
ServerSocket serverSocket = new ServerSocket(10477);
Socket socket = serverSocket.accept();
```

- `close` schließt den Port

Kommunikationssystem - Ein- / Ausgabe über Sockets

- Lesen von einem Socket mittels InputStream:

```
InputStream inStream = socket.getInputStream();
```

- Schreiben auf einen Socket mittels OutputStream:

```
OutputStream outStream = socket.getOutputStream();
```

- aus diesen Strömen können leistungsfähigere Ströme erzeugt werden:

```
VSObjectOutputStream VSOutStream =
    new VSObjectOutputStream(socket.getOutputStream());
```

Serverseite

Klasse `VSRemoteObjectManager`:

- Export von Objekten
- Erstellen von Remote-Referenzen für exportierte Objekte
- Aufruf von Methoden an exportierten Objekten
  - Suche des Objekts anhand der Objekt-ID
  - Suche der Interface-Methode über den generischen Methodennamen (Iteration über alle Methoden aller Interfaces und Stringvergleich des generischen Namens)
  - Aufruf der gefundenen Methode mit den übergebenen Parametern
  - Rückgabe des Rückgabewerts der aufgerufenen Methode

Klasse `VSServer`:

- Annahme von Verbindungen auf einem Port und Aufruf der richtigen `VSRemoteObjectManager`-Methode, je nach Anfrage
- Export von Objekten (durch `VSRemoteObjectManager`)

Serverseite - RemoteReference

Die von `VSRemoteObjectManager.getRemoteReference` zurückgegeben `VSRemoteReference` muss mindestens folgende Daten enthalten:

```
int objectID;
String host;
int port;
```

- `objectID`: Objekt-ID, über die auf das entsprechende Serverobjekt zugegriffen werden kann. Diese wird für den `invokeMethod`-Aufruf benötigt.
- `host`: Hostname des Serverrechners, aus dem `InetSocketAddress`-Objekt des Servers
- `port`: Serverport, aus dem `InetSocketAddress`-Objekt des Servers

Clientseite

Klasse `VSClient`:

- Suchen von Remote-Objekten, welche ein angegebenes Interface implementieren. Der Rückgabewert ist ein dynamischer Proxy für das Interface, dessen `invoke`-Methode die Informationen der Remote-Referenz auswertet.

Klasse `VSInvocationHandler`:

- Aufbau der Serververbindung
- Übertragen von Objekt-ID, generischem Methodennamen (`method.toGenericString()`) und Aufrufparametern
- Auswerten der Antwort des Servers und Rückgabe des empfangenen Rückgabewerts