Verteilte Systeme – Übung

Tobias Distler, Michael Gernoth

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

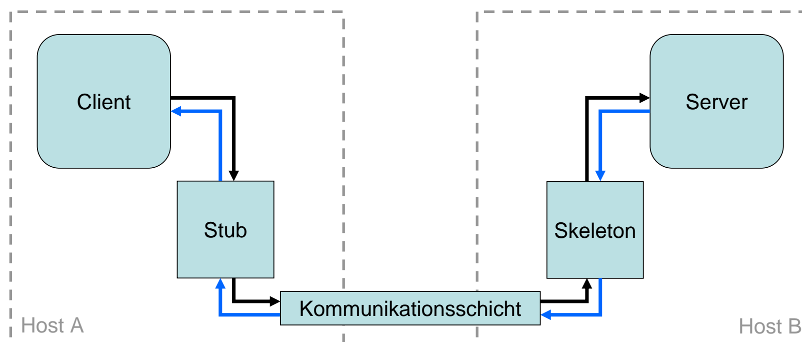
Sommersemester 2010

Überblick

Dynamische Proxies
Stubs & Skeletons
Dynamische Proxies als Stubs
Java Reflection für Skeletons
Übungsaufgabe 3

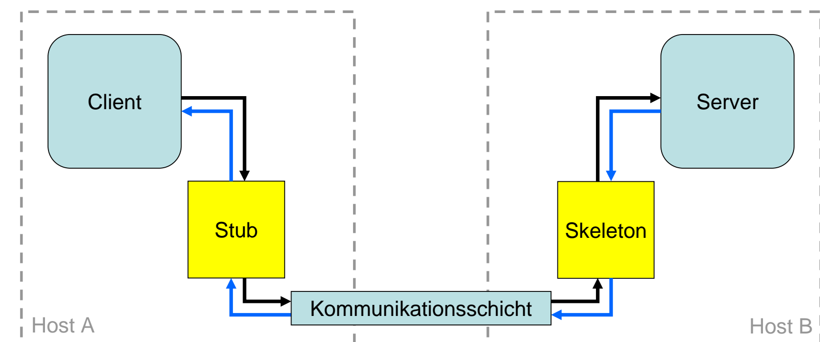
VS-Übung

- Entwicklung eines eigenen Fernaufrufsystems
- Orientierung an Java-RMI



Übungsaufgabe 3

- Erweiterung des Kommunikationssystems
- Dynamische Erzeugung von Stub und Skeleton



Stubs & Skeletons

- Grundprinzip eines (Methoden-)Fernaufrufs
  - Umsetzung von Methodenaufruf in Nachrichtenaustausch
  - Dabei Abstraktion der Örtlichkeit des
    - Auftragnehmers → durch den Stub
    - Auftraggebers → durch den Skeleton
  - Erfordert Ver-/Entpacken von Parametern/Rückgabewerten in Nachrichten
- Methodenstümpfe
  - Kapselung der obigen Aufgaben in Client-Stub und Server-Skeleton (bzw. Server-Stub)
  - Ziel: Automatische Erzeugung der Stümpfe

Lösungen in realen Systemen

- SUN-RPC
  - Explizite Schnittstellen-Beschreibungssprache
  - Eingabeparameter „by value", Rückgabewert „by result"
- Java-RMI
  - Parameterart direkt aus der Sprache ableitbar
  - Eingabeparameter
    - „by (object) reference", falls Remote-Objekt (→ Übungsaufgabe 4)
    - „by value", falls `Serializable`
- CORBA
  - Explizite Schnittstellen-Beschreibungssprache
  - Explizite Spezifikation von
    - „by value" (in)
    - „by result" (out)
    - „by value/result" (in/out)
  - Parameter vom Typ `interface` als entfernte Referenzen

Dynamische Proxies
Stubs & Skeletons
Dynamische Proxies als Stubs
Java Reflection für Skeletons
Übungsaufgabe 3

Stubs — Übersicht

- Stellvertreter des entfernten Objekts beim Aufrufer einer Methode → Bereitstellung einer Implementierung der **Schnittstelle des entfernten Objekts**
- Zentrale Aufgabe: Umwandlung eines lokalen Methodenaufrufs am Stub in einen Fernaufruf
  - Erzeugung einer Anfragenachricht
    - Eindeutige Kennung des Server-Prozesses
    - Eindeutige Kennung des entfernten Objekts
    - Eindeutige Kennung der aufzurufenden Methode
    - Einpacken der Aufrufparameter
  - Senden der Anfragenachricht über das Kommunikationssystem
  - Empfang einer Antwortnachricht per Kommunikationssystem
  - Auspacken des Rückgabewerts
  - Übergabe des Rückgabewerts an den Aufrufer

Beispiel — Schnittstelle 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);
    }
}
```


Manuelle Implementierung von Stubs

- Stub für Beispiel-Schnittstelle

```
public class HelloStub implements HelloInterface {
    public void setName(String name) {
        // Anfrage (IDs, Parameter) erstellen und senden
        // Fuer synchronen Aufruf: Antwort empfangen
    }
    public String getName() {
        // Anfrage (IDs) erstellen und senden
        String s = [...] // Antwort empfangen u. auspacken
        return s;
    }
    public void sayHello() {
        // Anfrage (IDs) erstellen und senden
        // Fuer synchronen Aufruf: Antwort empfangen
    }
}
```

- Nachteile
  - Hoher Implementierungsaufwand (v. a. bei Interface-Änderungen)
  - Code-Duplikation
  - Fehleranfällig

Dynamische Proxies — Übersicht

- Grundidee
  - Zur Laufzeit generierte Stellvertreterobjekte
  - Konfigurierbare Schnittstellen
  - Umleitung von Methodenaufrufen am Proxy auf einen zuvor registrierten **Invocation-Handler**
  - Anwendungsspezifische Implementierung des Invocation-Handlers
  - Weiterführende Dokumentation

`http://java.sun.com/javase/6/docs/technotes/guides/reflection/proxy.html`

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

- Dynamische Proxies als Stubs
  - Implementierung beliebiger Schnittstellen → Proxy kann als Stellvertreterobjekt für das entfernte Objekt dienen
  - Abfangen von lokalen Methodenaufrufen → Umwandlung in Fernaufrufe möglich

Dynamische Proxies — Invocation-Handler

- Implementierung eines Invocation-Handlers
  - Bereitstellung einer `invoke()`-Methode, an die sämtliche am Proxy getätigten Methodenaufrufe deligiert werden
  - Informationen über den ursprünglichen Aufruf (Methode, Parameter) bleiben dabei erhalten
  - Rückgabewert von `invoke()` wird als Rückgabewert des ursprünglichen Aufrufs verwendet
- Schnittstelle: `java.lang.reflect.InvocationHandler`

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

  - `proxy`: Der Proxy, an dem die `invoke`-Methode aufgerufen wurde
  - `method`: Das `Method`-Objekt der aufgerufenen Proxy-Methode
  - `args`: Array mit den Parametern des ursprünglichen Aufrufs
    (Falls kein Parameter übergeben wurde ist `args == null`, ansonsten kann die Anzahl der Parameter über `args.length` bestimmt werden.)

Dynamische Proxies — Erzeugung

- Proxy-Erzeugung mittels `Proxy.newProxyInstance()`

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

  - `loader`: Class-Loader für die Proxy-Klasse
    (Typischerweise der Class-Loader der zu implementierenden Schnittstelle; dieser kann durch den Aufruf von `getClassLoader()` am `Class`-Objekt der Schnittstelle bestimmt werden.)
  - `interfaces`: Array der zu implementierenden Interface-Klassen
  - `handler`: Instanz des Invocation-Handlers

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

Beispiel — Invocation-Handler

Invocation-Handler für **lokalen** Aufruf

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

public class HelloInvHandler implements InvocationHandler {
    private HelloInterface hello;

    public HelloInvHandler(HelloInterface hello) {
        this.hello = hello;
    }

    public Object invoke(Object proxy, Method m,
                         Object[] args) throws Throwable {
        System.out.println("[Proxy] Methode: " + m.getName());
        if(args != null) {
            System.out.println("[Proxy] Args: " + args.length);
        }
        return m.invoke(hello, args);   // eigentlicher Aufruf
    }
}
```


Beispiel — Proxy-Erzeugung und -Benutzung

Main-Methode (der Klasse `HelloTest`) zum Testen des Proxy

```
public static void main(String[] args) {
    // Erstellung des eigentlichen Objekts
    HelloInterface hello = new HelloImpl();

    // Erzeugung eines Invocation-Handlers
    HelloInvHandler handler = new HelloInvHandler(hello);

    // Proxy-Erzeugung
    ClassLoader loader = HelloInterface.class.getClassLoader();
    Class[] interfaces = new Class[] { HelloInterface.class };
    HelloInterface helloProxy = (HelloInterface)
           Proxy.newProxyInstance(loader, interfaces, handler);

    // Methodenaufrufe am Proxy
    helloProxy.setName("Benutzer");
    helloProxy.sayHello();
    System.out.println(helloProxy.getName());
}
```


Beispiel — Ausführung

- Beispiel-Ausführung

```
> java HelloTest
[Proxy] Methode: setName
[Proxy] Args: 1
[Proxy] Methode: sayHello
Hallo Benutzer
[Proxy] Methode: getName
Benutzer
```

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

Dynamische Proxies
Stubs & Skeletons
Dynamische Proxies als Stubs
Java Reflection für Skeletons
Übungsaufgabe 3

Skeletons — Übersicht

- Stellvertreter des Aufrufers einer Methode beim eigentlichen Objekt → Nachbildung des Verhaltens eines lokalen Aufrufers
- Zentrale Aufgabe: Ausführung des eigentlichen Methodenaufrufs
  - Empfang einer Anfragenachricht per Kommunikationssystem
    - Auspacken der Kennung des (jetzt lokalen) Objekts
    - Auspacken der Kennung der aufzurufenden Methode
    - Auspacken der Aufrufparameter
  - Bestimmung des Objekts mittels Kennung
  - Methodenaufruf am Objekt
  - Erzeugung einer Antwortnachricht mit dem Rückgabewert
  - Senden der Antwortnachricht über das Kommunikationssystem

Methodenaufrufe mit der Java Reflection API

- Bekannt aus Tafelübung 2
  - Aufruf mittels Reflection API (`java.lang.reflect.Method`)

```
Object invoke(Object obj, Object... args);
```

  - Beispiel

```
Method m = hello.getClass().getMethod("getName",
                                      new Class[0]);
String s = m.invoke(hello, (Object[]) null);
```

  - **Achtung:** Nicht zu verwechseln mit der `invoke()`-Methode des Proxy-Invocation-Handlers
- Problemstellung: Wie finde ich die richtige Methode?
  - Methodenname nicht eindeutig
  - Parameteranzahl nicht eindeutig
  - → Eindeutige Kennung muss den Methodennamen sowie Anzahl und Typen sämtlicher Parameter enthalten

Auswahl der richtigen Methode — `Method.toGenericString()`

- Eindeutige Kennung mittels `Method.toGenericString()`

```
public String toGenericString()
```

- Beispiel

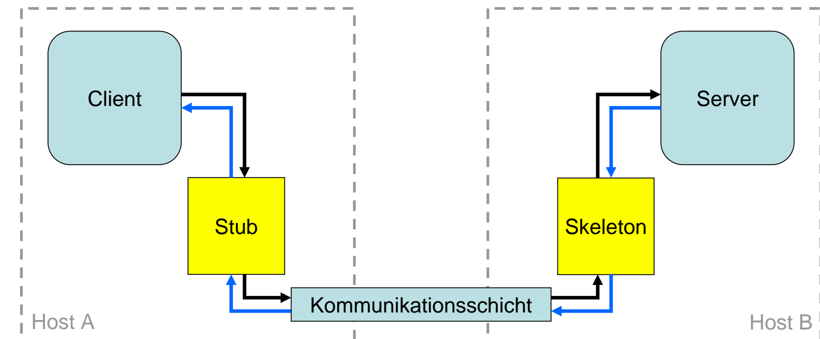
```
public abstract void HelloInterface.setName(java.lang.String)
```

  - Sichtbarkeit
  - Rückgabewert
  - Name der Methode mit Interface
  - Parameter
- Bestimmung und Verwendung des richtigen `Method`-Objekts
  - Abfrage aller Remote-Schnittstellen des Remote-Objekts
  - Abfrage aller Methoden dieser Schnittstellen
  - Vergleich der generischen Methoden-Strings mit übergebenem
  - Aufruf von `invoke()` am gefundenen Methoden-Objekt

Dynamische Proxies
Stubs & Skeletons
Dynamische Proxies als Stubs
Java Reflection für Skeletons
Übungsaufgabe 3

Übungsaufgabe 3

- Erweiterung des Kommunikationssystems
- Dynamische Erzeugung von Stub und Skeleton



Kommunikationssystem — Adressierung

- Eindeutige Identifikation eines Knotens im Fernaufrufsystem
  - IP-Adresse
    - DNS-Form, z. B. `www4.informatik.uni-erlangen.de`
    - durch Punkte getrenntes Quadtupel, z. B. `131.188.34.200`
  - Port-Nummer
- Java-Hilfsklasse: `java.net.InetSocketAddress`
  - Konstruktoren

```
public InetSocketAddress(String hostName, int port);
public InetSocketAddress(InetAddress addr, int port);
[...]
```

  - Beispiel

```
InetSocketAddress inetSockAddr = new InetSocketAddress
            ("www4.informatik.uni-erlangen.de", 10477);
```


Kommunikationssystem — TCP-Sockets

- Verwendung von TCP-Sockets
  - Kommunikationsendpunkt bei Client und Server
  - Zuverlässige Kommunikation
- Java-Klassen: `java.net.Socket` und `java.net.ServerSocket`
  - Verbindungsaufbau
    - Client-Seite

```
Socket socket = new Socket();
socket.connect(inetSockAddr);
```

    - Server-Seite

```
ServerSocket srvSocket = new ServerSocket(10477);
Socket socket = srvSocket.accept(); // blockierend
```

  - Verbindungsabbau

```
socket.close();
```

Kommunikationssystem — Ein-/Ausgabe über Sockets

- Einfache Ströme
  - Lesen und Schreiben einzelner Bytes bzw. von Byte-Arrays
  - Lesen von einem Socket mittels `java.io.InputStream`

    ```
    InputStream is = socket.getInputStream();
    ```
  - Schreiben auf einen Socket mittels `java.io.OutputStream`

    ```
    OutputStream os = socket.getOutputStream();
    ```
- Leistungsfähigere Ströme
  - Intern: Verwendung der einfachen Ströme
  - Lesen und Schreiben ganzer Objekte
  - Empfang von Objekten mittels `VSObjectInputStream`

    ```
    VSObjectInputStream ois = new VSObjectInputStream(is);
    ```
  - Senden von Objekten mittels `VSObjectOutputStream`

    ```
    VSObjectOutputStream oos = new VSObjectOutputStream(os);
    ```


Server-Seite

- Verwaltung von Remote-Objekten: `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 ihren generischen Namen
    - Aufruf der gefundenen Methode mit den übergebenen Parametern
    - Rückgabe des Rückgabewerts der aufgerufenen Methode
- Kommunikationssystem: `VSCommunication`
  - Annahme von Verbindungen
  - Empfangen von Anfragen, Senden von Antworten
- Bindeglied zwischen `VSRemoteObjectManager` und Kommunikationssystem: `VSServer`

Server-Seite — Remote-Referenzen

- Remote-Referenzen: `VSRemoteReference`

```
public class VSRemoteReference implements Serializable {
    private String host;
    private int port;
    private int objectID;
}
```

  - `host`: Host-Name des Servers
  - `port`: Nummer des Ports, auf dem das Kommunikationssystem des Servers Verbindungen annimmt
  - `objectID`: Objekt-ID, über die auf das entsprechende Remote-Objekt zugegriffen werden kann

Client-Seite

- Erzeugung von Stubs: `VSClient`
  - Lokale Lookup-Anfrage nach einer Schnittstelle
  - Beschaffung einer geeigneten Remote-Referenz vom Server
  - Erzeugung eines dynamischen Proxy aus der Remote-Referenz
  - Rückgabe des Proxy
- Abfangen lokaler Aufrufe am Proxy: `VSInvocationHandler`
  - Aufbau der Server-Verbindung
  - Zusammenstellung und Senden der Anfrage
    - Objekt-ID
    - (generischer) Methodenname (→ `Method.toGenericString()`)
    - Aufrufparameter
  - Empfang und Auswertung der Antwort des Servers
  - Rückgabe des empfangenen Rückgabewerts
- Kommunikationssystem: `VSCommunication`