Praktische Übung 2 zur Vorlesung “Verteilte Systeme” ETH Zürich, WS 2002/2003*

Prof. Dr. F. Mattern

Ausgabedatum: 6. Dezember 2002
Abgabedatum: 13. Januar 2003

Nachdem Sie in Aufgabe 1 selbst ein einfaches RPC-System implementiert haben, geht es in dieser Aufgabe um die Verwendung von RMI (*Remote Method Invocation*, dem von Java bereitgestellten RPC-Mechanismus) und CORBA (*Common Object Request Broker Architecture*, ein programmiersprachenunabhängiges RPC-System). Sie werden dazu ein erweitertes Bankenszenario einmal mittels RMI und einmal mittels CORBA implementieren.

Einführung

Im Vergleich zu Aufgabe 1 wurde das in dieser Aufgabe zu implementierende Bankenszenario um eine Bank und verschiedene Konto-Typen (Sparkonto und Girokonto) erweitert. Die Bank dient dazu, Konten zu verwalten (Erzeugen, Löschen, Suchen). Ein Sparkonto wird verzinst und verfügt daher über einen Zinssatz. Girokonten verfügen über ein Kreditlimit, das bei einer Abhebung nicht überschritten werden darf. Da beide Konto-Typen eine gemeinsame Funktionalität besitzen (nämlich Einzahlen, Abheben und Kontostandabfrage), gibt es eine gemeinsame Basisklasse `Konto`, von der `SparKonto` und `GiroKonto` erben. Die Bank verfügt über eine Methode `verzinse()`, die alle Sparkonten entsprechend ihrem Zinsatz verzinst, sowie Methoden `erzeuge_giro_konto()` und `erzeuge_spar_konto()` zum Erzeugen von Konten. Die letztgenannten Methoden sollen eine Exception auslösen, wenn die angegebene Kontonummer schon in Verwendung ist. Alle benötigten Klassen sollen in einer Package `Banking` untergebracht werden. Die Klassen sollen folgende Schnittstellen aufweisen:

```
package Banking;

public class BankException extends Throwable {
   ...
};

public interface Konto {
    public String konto_nummer();
    public void abheben (long betrag)
       throws BankException;
    public void einzahlen (long betrag);
```

*Die Abnahme dieser Aufgabe erfolgt am 13.1.2003 von 8.15 Uhr bis 10 Uhr in den Räumen IFW D31/D35. Danach wird es von 10 Uhr bis 11 Uhr in IFW A36 noch eine Nachbesprechung geben.

```
    public long kontostand ();
};

public interface GiroKonto extends Konto {
    public long get_kredit_limit ();
    public void set_kredit_limit (long limit);
};

public interface SparKonto extends Konto {
    public float get_zins_satz ();
    public void set_zins_satz (float zinssatz);
};

public interface Bank {
    public GiroKonto erzeuge_giro_konto (String konto_nummer, long kredit_limit)
        throws BankException;
    public SparKonto erzeuge_spar_konto (String konto_nummer, float zins_satz)
        throws BankException;
    public Konto finde (String konto_nummer);
    public void loesche (String konto_nummer);
    public void verzinsen ();
};
```

Die entsprechenden Dateien und weitere Code-Gerüste können Sie von der Vorlesungs-Webseite unter `http://www.inf.ethz.ch/vs/edu/WS0203/VS/`beziehen.

Teil 1 (RMI)

Implementieren Sie das obige Banken-Szenario mittels Java RMI. Sie können dazu ähnlich wie in Aufgabe 1 vorgehen und zunächst alle Klassen implementieren und mittels eines einfachen Testprogrammes auf Korrektheit überprüfen. Spalten Sie dann das bereits existierende Testprogramm in einen Client und einen Server auf. Der Server sollte zu Beginn ein Bank-Objekt instanziieren, auf das der Client dann mittels RMI zugreift, um neue Konten zu erzeugen und Transaktionen auf diesen Konten zu tätigen. Beachten Sie, dass sich die neu erzeugten Konto-Objekte ausschliesslich im Server befinden sollen.

Teil 2 (CORBA)

Während Java-RMI speziell auf die Programmiersprache Java zugeschnitten ist, handelt es sich bei CORBA um eine programmiersprachenunabhängige Middleware. Mit CORBA ist es also möglich, in verschiedenen Programmiersprachen implementierte Programme über Rechnergrenzen hinweg miteinander kommunizieren zu lassen. Sie werden diese Eigenschaft kennenlernen, indem Sie das Banken-Szenario mittels CORBA re-implementieren, und zwar den Server in Java und den Client in C++.

Vorbereitungen

Zunächst müssen Sie sicherstellen, dass Sie Java 1.2 verwenden. Geben Sie dazu auf Ihrem RIF/RAF-Account folgende Kommandos ein:

```
module addinit java/jdk/1.2.1
module add java/jdk/1.2.1
```

Den Java-Stub-Compiler finden Sie unter `/home/corba0/idltojava`.

Ausserdem müssen Sie sicherstellen, dass der richtige C++-Compiler verwendet wird, indem Sie folgende Kommandos ausführen:

```
module addinit gcc/2.95.3/pb
module add gcc/2.95.3/pb
```

Als C++-CORBA-Implementierung wird MICO[1] verwendet. Um die Pfade für die Verwendung von MICO richtig zu setzen, müssen Sie folgendes Kommando ausführen:

```
source /home/corba0/MICO/lib/mico-setup.csh
```

Beachten Sie, dass Sie das in jedem Terminal-Fenster und bei jedem Login erneut tun müssen. Um sich diesen Aufwand zu sparen, können Sie das Kommando auch am Ende Ihrer `.cshrc`-Datei einfügen.

Aufgabenstellung

Wie in der Vorlesung erklärt wird, verläuft die Entwicklung einer CORBA-Applikation ähnlich wie die Entwicklung einer Java-RMI-Applikation. Es gibt jedoch einen wichtigen Unterschied: um Programmiersprachenunabhängigkeit zu erreichen, müssen die Schnittstellen der durch CORBA zugreifbaren Objekte in einer speziellen Sprache, der sogenannten *IDL* (Interface Definition Language), definiert werden. Aus dieser Beschreibung generiert der Stub-Compiler später Client- und Server-Stubs in der verwendeten Programmiersprache. Wir stellen Ihnen dafür eine Datei `bank.idl` zur Verfügung, die die Schnittstellen des Bankenszenarios in IDL spezifiziert.

Implementieren Sie zunächst die Server-Seite in Java, d.h. die Klassen `BankImpl`, `KontoImpl`, `GiroKontoImpl` und `SparKontoImpl`, sowie ein Hauptprogramm, das eine Bank-Instanz erzeugt, beim ORB registriert und die Objektrefrenz in einer Datei `bank.ref` abspeichert.

Implementieren Sie dann den Klienten in C++, der die Objektreferenz der Bank aus der Datei `bank.ref` liest, in eine Referenz auf die entfernte Bank umwandelt und Konten erzeugt, Transaktionen durchführt etc. Wir stellen Ihnen ein `Makefile` bereit, d.h. Sie müssen nur noch eine Datei `client.cc` erstellen, die das Client-Programm enthält. Durch Eingabe von `make` können Sie dann ein ausführbares Programm `client` erzeugen.

Details zur Java-basierten CORBA-Implementierung finden Sie unter `http://java.sun.com/products/jdk/1.2/docs/api/overview-summary.html`, näheres zur C++-basierten CORBA-Implementierung MICO finden Sie unter `http://www.mico.org/doc/doc.html`.

[1] Siehe `www.mico.org`