Praktische Übung 3 zur Vorlesung "Verteilte Systeme" ETH Zürich, SS 2001

Prof. Dr. F. Mattern

Ausgabedatum: 8. Juni 2001
Abgabedatum 29. Juni 2001

In Aufgabe 2 haben Sie Java-RMI, eine Java-spezifische Middleware, kennengelernt. In dieser Aufgabe werden Sie *CORBA* (Common Objekt Request Broker Architecture) verwenden, um das schon aus Aufgabe 1 bekannte Banken-Szenario in etwas erweiterter Form zu implementieren.

Einführung

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 bereits aus Aufgabe 1 bekannte Banken-Szenario in etwas erweiterter Form implementieren und zwar den Server in Java und den Klienten in C++.

Wie in der Vorlesung noch 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.

Die IDL des zu implemtierenden Bankenszenarios lautet folgendermassen:

```
module Banking {
    exception BankException {
        string error;
    };

    interface Konto {
        string konto_nummer();
        void abheben (in unsigned long betrag)
            raises (BankException);
        void einzahlen (in unsigned long betrag);
        unsigned long kontostand ();
    };

    interface GiroKonto :  Konto {
        attribute unsigned long kredit_limit;
    };
```

```
    interface SparKonto : Konto {
        attribute float zins_satz;
    };

    interface Bank {
        GiroKonto erzeuge_giro_konto (in string konto_nummer,
                                      in unsigned long kredit_limit)
            raises (BankException);
        SparKonto erzeuge_spar_konto (in string konto_nummer,
                                      in float zins_satz)
            raises (BankException);
        Konto finde (in string konto_nummer);
        void loesche (in string konto_nummer);
        void verzinsen ();
    };
};
```

Im Unterschied zu Aufgabe 1 gibt es nun zwei Typen von Konten: Sparkonten und Girokonten. 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.

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 egcs/1.1.2
module add egcs/1.1.2
```

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.

[1] Siehe `www.mico.org`

Aufgabenstellung

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 Klienten-Programm enthält.

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.inf.ethz.ch/~mico/doc/doc.html`.