Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Institut für Informationssysteme, Dr. Markus Tresch

Uwe Röhm

Verteilte Objektverwaltung: Client, Server & Middleware
Lösungsblatt 4

Aufgabe 2: Implementierung verteilter Objekte mittels CORBA

Das Programm `uebung4.cc` musste wie folgt ergänzt werden. Hier ist nur der entscheidende Teil des Programms aus der `main()` Funktion mit der Eingabeschleife abgedruckt.

```
switch(choice)
{
   case 0 : // Kontoeröffnung
       try
       {
          Office->eroeffnen(Kunde);
          cout << "  Ihre neue Kontonummer lautet: " << Kunde->Konto() << endl;
       }
       catch ( Bank::OfficeClient::Reject &ex )
       {  cerr << "Kontoeröffnung nicht möglich: " << ex.Reason << endl; }
       catch (...)
       {  cerr << "Unerwartete Exception bei Kontoeröffnung!"   << endl; }
       break;


   case 1 : // Einzahlung
       try
       {
          CORBA::Float betrag;

          cout << "Wieviel Geld wollen Sie einzahlen?" << endl;
          cout << "> ";
          cin >> betrag;

          Office->einzahlen(Kunde->Konto(), betrag);
          cout << "  In Ordnung." << endl;
       }
       catch ( Bank::OfficeClient::Reject &ex )
       {  cerr << "Einzahlen nicht möglich: " << ex.Reason << endl; }
       catch (...)
       {  cerr << "Unerwartete Exception beim Einzahlen!"  << endl; }
       break;

   case 2 : // Abheben
       try
       {
          CORBA::Float betrag;
```

```
            cout << "Wieviel Geld wollen Sie abheben?" << endl;
            cout << "> ";
            cin >> betrag;

            Office->abheben(Kunde->Konto(), betrag);
            cout << "  In Ordnung." << endl;
        }
        catch ( Bank::OfficeClient::Reject &ex )
        { cerr << "Abheben nicht möglich: " << ex.Reason << endl; }
        catch (...)
        { cerr << "Unerwartete Exception bei Kontoeröffnung!" << endl;  }
        break;

    case 3 : // Kontostand abfragen
        try
        {
            cout << "  Sie haben momentan " << Office->kontostand( Kunde )
                 << " Franken auf Ihrem Konto." << endl;
        }
        catch ( Bank::OfficeClient::Reject &ex )
        { cerr << "Kontostandabfrage nicht möglich: " << ex.Reason << endl; }
        catch (...)
        { cerr << "Unerwartete Exception bei Kontostandsabfrage!"  << endl; }
        break;

    case 4 : // Adressenänderung
        try
        {
            char strasse[MAX_LEN];
            char land[MAX_LEN];
            char ort[MAX_LEN];

            cout << "In welcher Strasse wohnen Sie? ";
            cin.getline(strasse, MAX_LEN);
            cout << "In welchem Ort wohnen Sie? ";
            cin.getline(ort, MAX_LEN);
            cout << "In welchem Land liegt " << ort << "? ";
            cin.getline(land, MAX_LEN);

            Office->neue_adr(Kunde, strasse, ort, land);
        }
        catch ( ... )
        { cerr << "Unerwartete Exception beim Ändern der Adresse!" << endl; }
        break;

    case 5 : // Programmende
        cout << "Auf Wiedersehen!" << endl;
        done = true;
        break;

} /* switch(choice) */
```

Aufgabe 3: Implementierung des ATM-Serverobjekts

Das Skelettprogramm `ATMClient.cc` musste um die drei Methoden eines ATMClient Objekts ergänzt werden. Die Signaturen dieser Methoden standen in der Header-Datei `ATMClient.cc`:

```
class ATMClient_Impl : public Bank::ATMClientBOAImpl
{
   CORBA::string Lokation;

   virtual CORBA::string Standort ( CORBA::Environment & env );
   virtual void          abheben  ( Bank::Konto_ptr      konto,
                                    CORBA::Float         betrag,
                                    CORBA::Environment & env );
   virtual CORBA::Float  kontostand(Bank::Kunde_ptr      kunde,
                                    CORBA::Environment & env );
};
```

Die eigentliche Implementierung orientiert sich sehr stark an derjenigen eines `OfficeClient`s:

```
   CORBA::string ATMClient_Impl::Standort ( CORBA::Environment & env )
   {  return this->Lokation;  }

   void ATMClient_Impl::abheben ( Bank::Konto_ptr      konto,
                                  CORBA::Float         betrag,
                                  CORBA::Environment & env)
   {
      cout << "ATMClient::abheben(" << konto << ", " << betrag << ")" << endl;

      if ( ! konto->_isNull() )
      {
         if ( betrag > konto->Betrag() )
            throw Reject("Kontostand zu niedrig!");

         konto->abheben(betrag);
      }
      else
         throw Reject("Kein Konto vorhanden!");
   }

   CORBA::Float ATMClient_Impl::kontostand ( Bank::Kunde_ptr     kunde,
                                             CORBA::Environment &env )
   {
      cout << "ATMClient::kontostand(" << kunde << ")" << endl;

      if ( ! kunde->_isNull() && !kunde->Konto()->_isNull() )
        return kunde->Konto()->Betrag();
      else
        return throw Reject("Kein Konto vorhanden!");
   }
```

Zusätzlich war im Hauptprogramm und in dem zugehörigen Testprogramm `atm.cc` die Servername in den Aufrufen von `CORBA::Orbix.impl_is_ready(...)`, bzw. `Bank::ATMClient::_bind(...)` anzupassen.

```
// IDL Beschreibung unserer Bank-Miniwelt
module Bank {

  // Vorwaerts-Deklararionen dreier Objekttypen, die in
  // den Signaturen einiger Methoden bereits benoetigt werden.
  interface Konto;
  interface Kunde;
  interface Adresse;

  #define MAX_LEN 100

  interface OfficeClient
  {
    exception Reject { string Reason; };

    readonly attribute string<MAX_LEN> Name;

    Kunde neuer_kunde ( in string n, in string s, in string o, in string l) raises (Reject);
    Konto eroeffnen   ( in Kunde kunde )                                    raises (Reject);
    void  einzahlen   ( in Konto konto, in float betrag )                   raises (Reject);
    void  abheben     ( in Konto konto, in float betrag )                   raises (Reject);
    float kontostand  ( in Kunde kunde )                                    raises (Reject);
    void  neue_adr    ( in Kunde kunde,in string s,in string o,in string l);
  };

  interface ATMClient
  {
    exception Reject { string Reason; };

    readonly attribute string<MAX_LEN> Standort;

    void     abheben   ( in Konto konto, in float betrag ) raises(Reject);
    float    kontostand( in Kunde kunde )                  raises(Reject);
  };

  interface Adresse
  {
    readonly attribute string<MAX_LEN>  Strasse;
    readonly attribute string<MAX_LEN>  Ort;
    readonly attribute string<MAX_LEN>  Staat;
  };

  typedef long ID;

  interface Kunde
  {
             attribute string<MAX_LEN>  Name;
             attribute Konto            Konto;
    readonly attribute Adresse          Adresse;

    void     set_adr ( in string ort, in string strasse, in string land );
  };
```

```
  interface Konto
  {
    exception Reject{};

    readonly attribute ID    Nummer;
    readonly attribute Kunde Inhaber;
    readonly attribute float Betrag;

    void  eroeffnen   ( in ID nummer, in Kunde inhaber, in float betrag );
    void  einzahlen   ( in float betrag ) raises (Reject);
    void  abheben     ( in float betrag ) raises (Reject);
  };

  typedef long TID;

  interface TransService
  {
    TID     begin  ();
    boolean commit ( in TID tid );
    boolean abort  ( in TID tid );
  };

  typedef string<MAX_LEN> Name;

  interface NameService
  {
    exception NotFound{};
    exception InvalidName{};
    exception AlreadyBound{};

    void   bind        ( in Name name, in string obj ) raises (InvalidName, AlreadyBound);
    void   bind_context( in Name name )                raises (InvalidName,AlreadyBound);
    string resolve     ( in Name name )                raises (NotFound, InvalidName);
    void   unbind      ( in Name name )                raises (NotFound, InvalidName);
  };
};
```