JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

CORBA

Systemtechnik Labor
4BHIT 2015/16, GruppeX

Thomas Fellner

Version 1
Begonnen am 29. April 2016
Beendet am 6. Mai 2016

Note:
Betreuer: M. Borko

Inhaltsverzeichnis

1 Einführung **1**

1.1 Ziele 1

1.2 Voraussetzungen 1

1.3 Aufgabenstellung 1

2 Ergebnisse **2**

2.1 CORBA 2

2.2 Konfiguration und Installation 2

2.3 Programm 3

1 Einführung

Verteilte Objekte haben bestimmte Grunderfordernisse, die mittels implementierten Middlewares leicht verwendet werden können. Das Verständnis hinter diesen Mechanismen ist aber notwendig, um funktionale Anforderungen entsprechend sicher und stabil implementieren zu können.

1.1 Ziele

Diese Übung gibt eine einfache Einführung in die Verwendung von verteilten Objekten mittels CORBA. Es wird speziell Augenmerk auf die Referenzverwaltung sowie Serialisierung von Objekten gelegt. Es soll dabei eine einfache verteilte Applikation in zwei unterschiedlichen Programmiersprachen implementiert werden.

1.2 Voraussetzungen

- Grundlagen Java, C++ oder anderen objektorientierten Programmiersprachen
- Grundlagen zu verteilten Systemen und Netzwerkverbindungen
- Grundlegendes Verständnis von nebenläufigen Prozessen

1.3 Aufgabenstellung

Verwenden Sie das Paket ORBacus [1] oder omniORB [2] bzw. JacORB [3] um Java und C++ ORB-Implementationen zum Laufen zu bringen. [4] [5]

Passen Sie eines der Demoprogramme (nicht Echo/HalloWelt) so an, dass Sie einen Namingservice verwenden, welches ein Objekt anbietet, das von jeweils einer anderen Sprache (Java/C++) verteilt angesprochen wird. Beachten Sie dabei, dass eine IDL-Implementierung vorhanden ist um die unterschiedlichen Sprachen abgleichen zu können.

Vorschlag: Verwenden Sie für die Implementierungsumgebung eine Linux-Distribution, da eine optionale Kompilierung einfacher zu konfigurieren ist.

2 Ergebnisse

Zu finden ist mein Beispiel auf Github unter `https://github.com/tfellner-tgm/SYT-rmi` [6]

2.1 CORBA

CORBA ermöglicht einem, Objekte über mehrere Prozesse und sogar durch das Netzwerk verwendbar zu machen. Im Gegensatz zu RMI ist dies aber Programmiersprachen unabhängig, doch das Prinzip ist sehr ähnlich.

2.2 Konfiguration und Installation

Hier ist erklärt wie man die 2 ORBs, JacORB für Java und OmniORB für C++ bzw. auch Python, aufsetzt.
Zuerst müssen die binaries von OmniORB und JacORB mittels z.B. `wget` installiert werden.

```
1 wget https://sourceforge.net/projects/omniorb/files/omniORB/omniORB-4.2.1/omniORB-4.2.1-2.tar.bz2/download
  wget http://www.jacorb.org/releases/3.7/jacorb-3.7-binary.zip
```

Listing 1: Download von OmniORB und JacORB

Um OmniORB zu installieren muss man ein build directory erstellen im Ordner des extrahierten tars.
Danach wird `../configure` ausgefürt in diesem build Ordner, um die nötigen `make` dateien zu generieren.
Um `make` ausführen zu können, müssen die Packages `build-essential`, für gcc, make ... und `libpython2.7-dev`, für Python, installiert werden.
Danach kann der `make` Befehl angewandt werden und um es dann noch zu installieren `make install`
Da Libraries hinzugefügt wurden, muss die `LD_LIBRARY_PATH` Variable noch geupdated werden mittels `ldconfig`
Da ich mit den OmniORB Namingservice arbeite muss dieser vor einem Test gestartet werden. Um ihn starten zu können muss aber noch der Ordner `/var/omninames` erstellt werden. Nun kann der Service mittels `omniNames -start -always` gestartet werden.

Für JacORB muss zuerst die `openjdk-8-jdk` installiert werden. Da wir binaries installiert haben kann man den ungezipten Folder in `/opt/` geben und dort dann ein Symlink zu der aktuellsten Version machen.

Zur Testung beider ORBS habe ich das halloWelt Beispiel [7] verwendet. Wobei hier die directories richtig gesetzt werden müssen (von mike zu dem Homeverzeichnis des Users) und dann der Client mittels `ant run-client` und der Server mittels `make run` gestartet werden.

2.3 Programm

Ich habe das Hallo Welt Beispiel [7] zu einem simplen Rechner umgeändert.

Im Vergleich zu RMI ist das `UnicastRemoteObject` hier das `orb` (Object Request Broker) Objekt, diese kümmert sich um den Austausch der Objekte. POA (Portable Object Adapter) leitet die Aufrufe weiter an die Konkrete Implementierung, damit die orb vom tatächlichen Code getrennt sind.

```
int main(int argc, char **argv)
{
  try {
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

    CORBA::Object_var obj = orb->resolve_initial_references("RootPOA");
    PortableServer::POA_var poa = PortableServer::POA::_narrow(obj);

    Calculation* calc = new Calculation();

    PortableServer::ObjectId_var calcid = poa->activate_object(calc);

    // Obtain a reference to the object, and register it in
    // the naming service.
    obj = calc->_this();

    CORBA::String_var x;
    x = orb->object_to_string(obj);
    cout << x << endl;

    if( !bindObjectToName(orb, obj) )
      return 1;

    calc->_remove_ref();

    PortableServer::POAManager_var pman = poa->the_POAManager();
    pman->activate();

    orb->run();
  }
  catch(CORBA::SystemException& ex) {
    cerr << "Caught CORBA::" << ex._name() << endl;
  }
  catch(CORBA::Exception& ex) {
    cerr << "Caught CORBA::Exception: " << ex._name() << endl;
  }
  catch(omniORB::fatalException& fe) {
    cerr << "Caught omniORB::fatalException:" << endl;
    cerr << "  file: " << fe.file() << endl;
    cerr << "  line: " << fe.line() << endl;
    cerr << "  mesg: " << fe.errmsg() << endl;
  }
  return 0;
}
```

Listing 2: server.cc main Klasse welches die ORB und POA Objekte erstellt

Hier ist dann zu sehen wie ein Objekt gebindet wird.

```
static CORBA::Boolean
bindObjectToName(CORBA::ORB_ptr orb, CORBA::Object_ptr objref)
{
  CosNaming::NamingContext_var rootContext;
  // Obtain a reference to the root context of the Name service:
  CORBA::Object_var obj;
  obj = orb->resolve_initial_references("NameService");

  // Narrow the reference returned.
  rootContext = CosNaming::NamingContext::_narrow(obj);

  try {
    // Bind a context called "test" to the root context:
    CosNaming::Name contextName;
    contextName.length(1);
    contextName[0].id   = (const char*) "test";       // string copied
    contextName[0].kind = (const char*) "my_context"; // string copied

    CosNaming::NamingContext_var testContext;
    try {
      // Bind the context to root.
      testContext = rootContext->bind_new_context(contextName);
    }
    catch(CosNaming::NamingContext::AlreadyBound& ex) {
      CORBA::Object_var obj;
      obj = rootContext->resolve(contextName);
      testContext = CosNaming::NamingContext::_narrow(obj);
    }

    // Bind objref with name Calculate to the testContext:
    CosNaming::Name objectName;
    objectName.length(1);
    objectName[0].id   = (const char*) "Calculation";   // string copied
    objectName[0].kind = (const char*) "Object"; // string copied

    try {
      testContext->bind(objectName, objref);
    }
    catch(CosNaming::NamingContext::AlreadyBound& ex) {
      testContext->rebind(objectName, objref);
    }
  }
  catch(CORBA::TRANSIENT& ex) {
    cerr << "Caught system exception TRANSIENT -- unable to contact the "
         << "naming service." << endl
         << "Make sure the naming server is running and that omniORB is "
         << "configured correctly." << endl;

    return 0;
  }
  catch(CORBA::SystemException& ex) {
    cerr << "Caught a CORBA::" << ex._name()
         << " while using the naming service." << endl;
    return 0;
  }
  return 1;
}
```

Listing 3: server.cc main Klasse welches die ORB und POA Objekte erstellt

Das IDL File wird verwendet um die Interfaces zu generieren, welche dann von Java und C++ verwendet werden.

```
#ifndef __CALCULATOR_IDL__
#define __CALCULATOR_IDL__
module calculator {
    exception DivisionThroughZero {
        string message;
    };

    interface Calculation {
        double add(in double nr1, in double nr2);
        double subtract(in double nr1, in double nr2);
        double multiply(in double nr1, in double nr2);
        double divide(in double nr1, in double nr2) raises (DivisionThroughZero);
    };
};
#endif  // __CALCULATOR_IDL__
```

Listing 4: calculator.idl Interface

Wie vorhin erwähnt braucht das Programm noch Konkrete Implementierungen dieses Interfaces.

```
class Calculation : public POA_calculator::Calculation
{
public:
    inline Calculation() {}
    virtual ~Calculation() {}
    virtual double add(const double nr1, const double nr2);
    virtual double subtract(const double nr1, const double nr2);
    virtual double multiply(const double nr1, const double nr2);
    virtual double divide(const double nr1, const double nr2);
};


double Calculation::divide(const double nr1, const double nr2) {
    if(nr2 == 0) {
        throw calculator::DivisionThroughZero("Division by zero is undefined");
    }

    return CORBA::Double (nr1 / nr2);
}

double Calculation::add(const double nr1, const double nr2) {
    return CORBA::Double (nr1 + nr2);
}

double Calculation::subtract(const double nr1, const double nr2) {
    return CORBA::Double (nr1 - nr2);
}

double Calculation::multiply(const double nr1, const double nr2) {
    return CORBA::Double (nr1 * nr2);
}
```

Listing 5: C++ IDL Implementation

Diese Methoden können dann von Java mittels dem hier erzeugtem Objekt ...

```
public static Calcultion connectToRemote(String[] args) {
    try {
        /* Erstellen und intialisieren des ORB */
        ORB orb = ORB.init(args, null);

        /* Erhalten des RootContext des angegebenen Namingservices */
        Object o = orb.resolve_initial_references("NameService");

        /* Verwenden von NamingContextExt */
        NamingContextExt rootContext = NamingContextExtHelper.narrow(o);

        /* Angeben des Pfades zum Calculate Objekt */
        NameComponent[] name = new NameComponent[2];
        name[0] = new NameComponent("test","my_context");
        name[1] = new NameComponent("Calculation", "Object");

        /* Aufloesen der Objektreferenzen  */
        return CalculationHelper.narrow(rootContext.resolve(name));

    } catch (Exception e) {
        System.err.println("Es ist ein Fehler aufgetreten: " + e.getMessage());
        e.printStackTrace();

        return null;
    }
}
```

Listing 6: Client.java JacORB Verbindung

... aufgerufen werden.

```
public static void main(String[] args)  {

    //create Calculator Object
    Calculation calculator = connectToRemote(args);
    System.out.println("5 + 2 = " + calculator.add(5, 2));
    System.out.println("2 - 3 = " + calculator.subtract(2, 3));
    System.out.println("2 * 4.35 = " + calculator.multiply(2, 4.35));

    try {
        System.out.println("5.2 / 2.5 = " + calculator.divide(5.2, 2.5));

        //Exception Erzeugen
        calculator.divide(1,0);
    }catch(DivisionThroughZero e) {
        System.err.println(e.message);
    }
}
```

Listing 7: Client.java JacORB Verbindung

Literatur

[1] Micro Focus. Orbacus. `https://www.microfocus.com/products/corba/orbacus/orbacus.aspx`.

[2] Duncan Grisby. Omniorb : Free corba orb. `http://omniorb.sourceforge.net/`, 28.09.2015.

[3] Jacorb - the free java implementation of the omg's corba standard. `http://www.jacorb.org/`, 03.11.2015.

[4] Duncan Grisby. The omniorb version 4.2 users' guide. `http://omniorb.sourceforge.net/omni42/omniORB.pdf`, 11.03.2014.

[5] Inc. IONA Technologies. Corba/c++ programming with orbacus student workbook. url-http://www.ing.iac.es/ docs/external/corba/book.pdf , 11.03.2014.

[6] Thomas Fellner. Github repository mit der aufgabe. `https://github.com/tfellner-tgm/CORBA`.

[7] Michal Borko. Corba hallo welt beispiel mit java und c++. `https://github.com/mborko/code-examples/tree/master/corba/halloWelt`.

Tabellenverzeichnis

Listings

1 Download von OmniORB und JacORB 2
2 server.cc main Klasse welches die ORB und POA Objekte erstellt 3
3 server.cc main Klasse welches die ORB und POA Objekte erstellt 4
4 calculator.idl Interface 5
5 C++ IDL Implementation 5
6 Client.java JacORB Verbindung 6
7 Client.java JacORB Verbindung 6

Abbildungsverzeichnis