JUST DO IT
HÖHERE ABTEILUNG FÜR
INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

Common Object Request Broker

Systemtechnik Labor
4BHIT 2015/16, GruppeB

Philip Vonderlind

Version 0.1
Begonnen am 20.5.2016
Beendet am 27.5.2016

Note:
Betreuer: Prof. Borko

Inhaltsverzeichnis

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 oder omniORB bzw. JacORB um Java und C++ ORB-Implementationen zum Laufen zu bringen.

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

2.1 CORBA Basics

CORBA ist kurz für 'Common Object Request Broker' und ermöglicht es Objekte für mehrere Prozesse in einem Netzwerk nutzbar zu machen. Dabei bleiben diese CORBA Objekte im Gegensatz zu 'Remote Method Invocation' nicht abhängig von einer Programmiersprache, es können also mehrere verwendet werden. (In unserem Beispiel Java und C++)
CORBA wird auch oft als eine Art 'Softwarebus' bezeichnet, was in dieser Graphik gut erkenntlich ist: Die Kommunikation zwischen den einzelnen Servern und Clients erfolgt über sogenannte 'ORB's

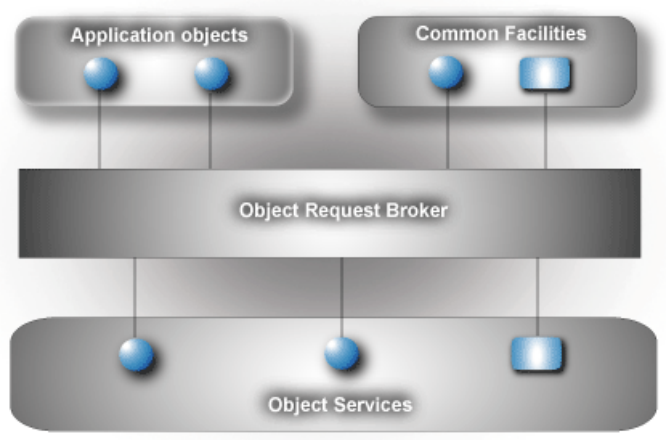

Abbildung 1: CORBA BUS "http://www.ois.com/Products/what-is-corba.html"

(Object Request Broker), welche die Location des Ziels beschreiben, einen Request an das Objekt senden und eine Response an den Aufrufenden zurückgeben.

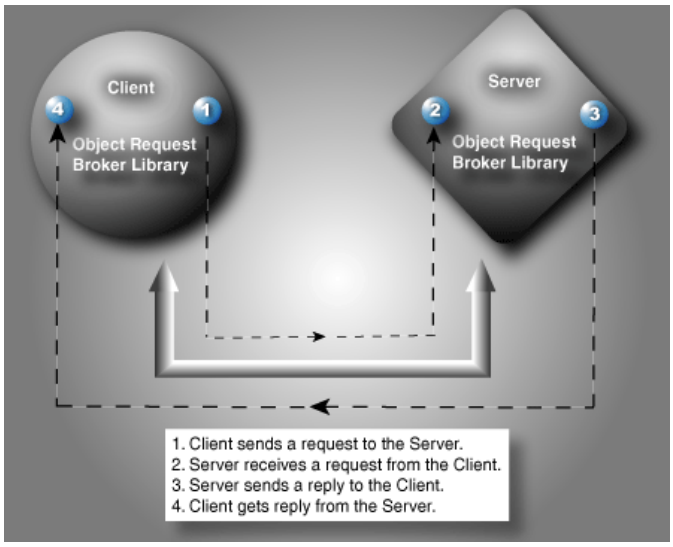

Abbildung 2: ORB "http://www.ois.com/Products/what-is-corba.html"

2.2 Installation der benötigten Packete

Um den Testcode ausführen zu können, benötigt man jeweils einen 'Object Request Broker' für C++ und Java. In unserem Fall haben wie hier OmniORB für C++ und JacORB für Java gewählt. Beide dieser Packete lädt man am besten mit wget wie folgt einfach herunter:

```
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: Herunterladen der Packete

2.2.1 OmniORB

In einem seperaten 'build' Verzeichniss, extrahiert man nun den omniORB-Tarball und führt man die Konfigurationsdatei mit ./configure im build-Ornder aus. Nun muss das Packet noch mit 'make' kompiliert und installiert werden. Hierzu ist aber das Python-Version 2.7 benötigt, welche man mit dem Packetmanager installieren kann (einfach libpython-dev installieren, die default Version ist 2.7). Falls man make und die build-essential noch nicht installiert hat, muss man dies auch noch nachholen. Nun kann man mit 'make && make install' omniORB installieren. Nun kann der Service auch schon mit 'omniNames -start -always' gestartet werden.

2.2.2 JacORB

Da man für JacORB die Java 8 Version benötigt muss diese noch extra installiert werden. Dies ist unter Debian etwas kompliziertet, da die 8er Version noch nicht in den offiziellen Repositories vorhanden ist. Um Java 8 zu installieren, bin ich folgendem Tutorial gefolgt: `"http://www.mkyong.com/java/how-to-install-oracle-jdk-8-on-debian/"`

2.3 Testen des Tutorialcodes

Um zu testen ob die Konfiguration funktioniert, öffnet man zwei Terminal Fenster und navigiert auf einem in den client Ordner des Testcodes und mit dem anderen in den server Ordner (der Tutorialcode wurde zuvor mit git clone `"https://github.com/mborko/code-examples"` aus dem bereitgestellten Repo heruntergeladen). Nun führt man am Server den Befehl 'make run' aus und dann am Client den Befehl 'ant run-client'.

2.4 Implementierung des eigenen Programms

2.5 Ändern von des Objectes auf ein Calculator Object

Damit die neuen Methoden verwendet werden können, muss dem POA auch das passende Objekt mitgegeben werden. Dazu ändert man in der server.cc das activate_object zu einem Calculator Objekt.

```
    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);
```

Listing 2: Anpassen des POA

2.5.1 Binden an den ORB

Das Object muss natürlich auch noch gebunden werden, damit man den Naming Context auch gefunden werden kann.

```
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
    // Note on kind: The kind field is used to indicate the type
    // of the object. This is to avoid conventions such as that used
    // by files (name.type -- e.g. test.ps = postscript etc.)

    CosNaming::NamingContext_var testContext;
    try {
      // Bind the context to root.
      testContext = rootContext->bind_new_context(contextName);
    }
    catch(CosNaming::NamingContext::AlreadyBound& ex) {
      // If the context already exists, this exception will be raised.
      // In this case, just resolve the name and assign testContext
      // to the object returned:
      CORBA::Object_var obj;
      obj = rootContext->resolve(contextName);
      testContext = CosNaming::NamingContext::_narrow(obj);
      if( CORBA::is_nil(testContext) ) {
        cerr << "Failed to narrow naming context." << endl;
        return 0;
      }
    }

    // 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);
    }
```

Listing 3: Binden an den ORB

2.5.2 ORB am Java Client erzeugen

Nachdem man den C++ ORB erstellt hat, muss man das selbe auch noch in Java machen. In der Main-Methode wird auch gleich der Callback des Servers auf die Console ausgeben. Dies funktioniert wie folgt:

```
    package calculator;

import org.omg.CosNaming.*;
import org.omg.CORBA.*;
import org.omg.CORBA.Object;

public class Client {

    public static void main(String[] args)  {

        //create Calculator Object
        Calculation calculator = connectToRemote(args);
        System.out.println("The greatest common divisor of 100 und 15 is "+calculator.ggt(100.0,15.0));
        System.out.println("The smallest common factor of 1550 and 15 is "+calculator.kgv(1550,15));
    }

    public static Calculation 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 4: ORB in Java implementieren

2.5.3 Funktionalität am Server

Damit die Aufrufe am Client auch funktionieren, muss die Funktionalität natürlich auch am Server vorhanden sein. Dazu fügt man im Contructor ein paar Methodenköpfe hinzu und implemetiert diese anschließend.

```
#include <calculator.hh>
#ifdef HAVE_STD
#  include <iostream>
   using namespace std;
#  include <math.h>
#else
#  include <iostream.h>
#endif

static CORBA::Boolean bindObjectToName(CORBA::ORB_ptr, CORBA::Object_ptr);
class Calculation : public POA_calculator::Calculation
{
public:
    inline Calculation() {}
    virtual ~Calculation() {}
    virtual double ggt(const double numer, const double denom);
    virtual double kgv(const double numer, const double denom);
};

double Calculation::kgv(const double numer, const double denom){
    double n1 = numer;
    double n2 = denom;
    double num = numer;
    double den = denom;
    double ggt = 0;

    if(num<0){  num=-num; }
    if(den<0){  den=-den; }
    int i = 0;
    while(i<1){
        if(num==0){ // trap if a==0
            ggt = den;
        i=1;
    }
     den = fmod(den,num); // otherwise here would be an error
        if(den==0){ // trap if b==0
            ggt = num;
        i=1;
    }
        num = fmod(num,den); // otherwise here would be an error
    }

    return CORBA::Double((n1 * n2) / ggt );
}

double Calculation::ggt(const double numer, const double denom) {
    double num = numer;
    double den = denom;

    if(num<0){ num=-num; }
    if(den<0){ den=-den; }
    for(;;){
        if(num==0) // trap if a==0
            return CORBA::Double(den);
        den = fmod(den,num); // otherwise here would be an error
        if(den==0) // trap if b==0
            return CORBA::Double(num);
        num = fmod(num,den); // otherwise here would be an error
    }

}
```

Listing 5: Funktionalität am Server hinzufügen

3 Quellen

1. **GitHub der Aufgabe** "https://github.com/pvonderlin-tgm/SytCorba"
2. **CORBA Introduction** "http://www.ois.com/Products/what-is-corba.html"
3. **CORBA C++ ORB** "http://omniorb.sourceforge.net/"
4. **CORBA Java ORB** "http://www.jacorb.org/"
5. **omniORB Userguide** "http://omniorb.sourceforge.net/omni42/omniORB.pdf"
6. **GGT und KGV Algorithmus** "https://www.c-plusplus.net/forum/23171-full"

4 Zeitaufzeichnung

Datum	Dauer	Beschreibung
20.5.2016	4 Stunden	Aufsetzen der virtuellen Maschinen, Testen des Codes
27.5.2016	2 Stunden	Eigene Methoden implementieren
6.6.2016	2 Stunden	Finalisieren von Protokoll und Programm

Listings

Abbildungsverzeichnis

