JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

CORBA

Systemtechnik Labor
4BHIT 2015/16, Gruppe X

Colakovic Zeljko

Version 1.0

Note:

Betreuer: M.BORKO

Begonnen am 29. April 2016

Beendet am 05. Mai 2016

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.

1.4 Quellen

[1] "omniORB : Free CORBA ORB"; Duncan Grisby; 28.09.2015; online: http://omniorb.sourceforge.net/

[2] "Orbacus"; Micro Focus; online: https://www.microfocus.com/products/corba/orbacus/orbacus.aspx

[3] "JacORB - The free Java implementation of the OMG's CORBA standard."; 03.11.2015; online: http://www.jacorb.org/

[4] "The omniORB version 4.2 Users' Guide"; Duncan Grisby; 11.03.2014; online: http://omniorb.sourceforge.net/omni42/omniORB.pdf

[5] "CORBA/C++ Programming with ORBacus Student Workbook"; IONA Technologies, Inc.; September 2001; online: http://www.ing.iac.es/~docs/external/corba/book.pdf

2 Ergebnisse

Folgende Schritte wurden auf einem Rechner mit Windows 10 durchgeführt.

Der vollständige Code ist auf github unter folgendem Link vorzufinden: *https://github.com/zcolakovic-tgm/SYT.git*

2.1 CORBA

CORBA ist eine erweiterte Form von RMI. Mit CORBA ist es möglich Objekte mit Verhalten und Eigenschaften in mehreren Programminstanzen darzustellen, auch übers Netzwerk hinaus und das unabhängig von der Programmiersprache.

2.2 Installation

Zuerst wird im Ordner das File */etc/apt/sources.list* gändert. Es wird alles hinausgelöscht bis auf folgende Zeilen:

- deb http://httpredir.debian.org/debian testing main
- deb-src http://httpredir.debian.org/debian testing main
- deb http://security.debian.org/ testing/updates main
- deb-src http://security.debian.org/ testing/updates main

Danach werden die entsprechenden Libraries runtergeladen mittels *wget*

- *https://sourceforge.net/projects/omniorb/fies/omniORB/omniORB-4.2.1/omniORB-4.2.1-2.tar.bz2/download*
- *http://www.jacorb.org/releases/3.7/jacorb-3.7-binary.zip*

Nun um OmniORB zu compilieren werden Tools benötigt wie *make.* Hierfür einfach mit *apt-get install build-essential* ausführen. Danach muss in OmniORB ein neuer Ordner erzeugt dort werden dann die Binarys abgelegt. Nun wird noch Python SDK 2.7 benötigt. Hier führ einfach *apt-get install libpython2-7-dev* ausführen. Nach dem Download kann man mittels *../configure* die benötigten Daten erstellen. Nun kann man mit *make* die Library installieren und mit einem *sudo make isntall* am System installieren. Nun wird noch dem System mitgeteilt das eine neue Library installiert wurde und das wird mit *ldconfig* getan.

Nun muss ein Ordner namens *omniNames* erstellt mittels *mkdir /var/omniNames*. Nun kann man den Dienst mit *sudo omniNames –start –always* ausführen.

Um Java-Files zu kompilieren wäre es vom Vorteil *sudo apt-get install openjdk-8-jdk openjdk-8-jre libatk-wrapper-java-jni libgtk-3-0 librest-0.7-0 libsoup2.4-1 glib-networking libgnutls30* auszuführen.

Um JacORB zu installieren ist es nötig ein *opt* Ordner im Home-Verzeichnis anzulegen. Danach führt man *ln –s <JacORB-Ordner> /opt/jacorb* aus.

Um es nun Lauffähig zu machen wird im *MAKEFILE* der Pfad zum *OMNIDL* auf das Kommando *omnidl* gändert.

Dazu muss im *build.xml* der Pfad zum *jacorb.dir* richtig gesetzt werden.

2.2.1 Zusammenfassung

- sudo apt-get install build essential
- wget <JackORB-binary><OmniORB 4.2.1-2>
- cd OmniORB
- mkdir build
- cd build
- sudo apt-get install libpython2.7-dev
- ../configure
- make
- sudo make install
- sudo ldconfig
- mkdir /var/omniNames
- sudo omniNames –start –always
- mkdir /opt
- cd opt
- ln –s <JacORB-Ordner> /jacorb

2.3 Code

2.3.1 Server

Im fogenden Code sind einige Variablen die für ein besseres Verständnis erklärt werden müssen. *ORB* steht für Object Request Broker und ist die Middleware zwischen Programm und den Austausch der Objekte handhabt. *POA* steht für Portable Object Adapter und ist dafür zuständig das mit Hilfe der Referenzen das nicht lokale Objekte aufzurufen.

```
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);

        Calculate* myecho = new Calculate();

        PortableServer::ObjectId_var myechoid = poa->activate_object(myecho);

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

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

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

        myecho->_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;
}
```

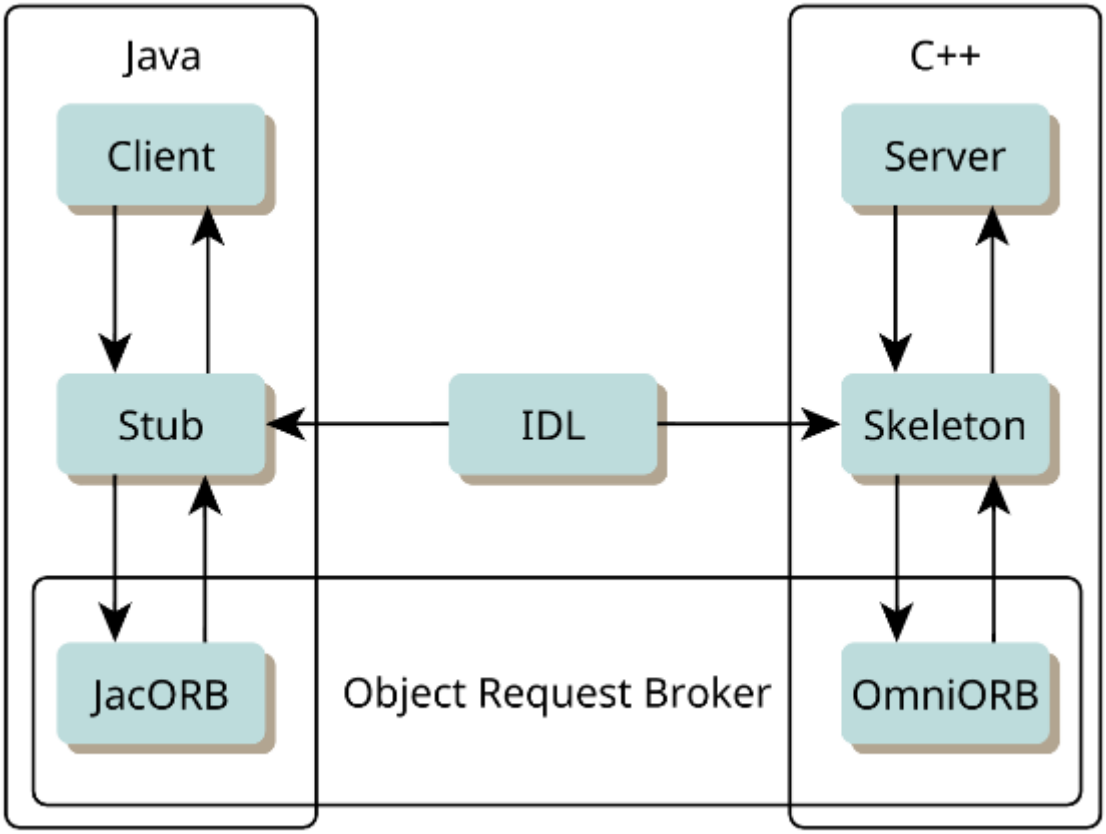
Der Server beinhaltet die Funktion der Methoden die vom Client abgerufen werden.

```
/**
* class Calculate
*
* Klasse die die Funktionalitaeten beinhaltet
*
* author Zeljko Colakovic
* date 04-05-2016
*/
class Calculate : public POA_calculator::Calculate
{
public:
    inline Calculate() {}
    virtual ~Calculate() {}
    virtual int div(const int a, const int b);
    virtual int add(const int a, const int b);
    virtual int sub(const int a, const int b);
    virtual int mult(const int a, const int b);
};


/**
* Addier zwtei Zahlen zusammen
* param a
* param b
* return Ergebnis
*/
int Calculate::add(const int a, const int b) {
    return CORBA::Long(a + b);
}
```

2.3.2 Idl

Das *idl-File* ist ein sprachunabhängiges Interface das als Schnittstelle als zwischen Stub und Skeletons dient.



Interface, Methoden des Rechners:

```
#ifndef __CALCULATOR_IDL__
#define __CALCULATOR_IDL__
module calculator {

    interface Calculate {
        long add(in long a, in long b);
        long sub(in long a, in long b);
        long mult(in long a, in long b);
        long div(in long a, in long b);
    };
};
#endif  // __CALCULATOR_IDL__
```

2.3.3 Client

Im Client sind nur *Wrapper-Klasse* inkludiert. Die Funktionalität wird am Server implementiert.

```
public class Client {
    public Client calculate = null;

    /**
     * Konstruktor
     *
     * @param args - Kommondozeilen Parameter
     * @since 04-05-2016
     */
    public Client(String[] args) {
        connectToRemote(args);
    }

    /**
     * Wrapper Methode welche die locale referenz der remote Methode aufruft
     *
     * @param a - Integer Eingabe
     * @param b - Integer Eingabe
     * @return - Integer Ergebnis
     * @since 04-05-2016
     */
    public int add(int a, int b) {
        return calculate.add(a, b);
    }
```

Hier wird die Verbindung zum Server hergestellt.

```
/**
 * Method welches die Verbingung erstellt
 *
 * @param args - CLI Argumente
 * @since 04-05-2016
 */
public void 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 Echo Objekt
        NameComponent[] name = new NameComponent[2];
        name[0] = new NameComponent("test","my_context");
        name[1] = new NameComponent("Berechne", "Object");

        // Aufloesen der Objektreferenzen
        calculate = CalculateHelper.narrow(rootContext.resolve(name));

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

2.4 Probleme

Client lässt sich mit *ant run-client* nicht *builden*. Als Fehlermeldung wird *BUILD FAILED /home/zeljko/Download/CORBA/client/build.xml:60: Compile failed; see the compile error output for details.* Wenn man nun ins *Client.java* geht wird die Zeile calculate = CalculateHelper.narrow(rootContext.resolve(name)); markiert. Die *Helper-Klasse* ist eine generierte Klasse wo sich der Fehler nicht befinden kann. In narrow wird der Fehler ebenfalls ausgeschloße da diese Methoden einwandfrei beim *HelloWorld-Example* funktioniert haben.

2.4 Zeitaufwand

Datum	Aufgabe	Zeit
Von 29.04.2016 Bis 05.05.2016	Tutorial	2 Stunden
Von 29.04.2016 Bis 05.05.2016	Erweiterung – Calculator	4 Stunden
Von 29.04.2016 Bis 05.05.2016	Protokoll	2 Stunden