JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

Verteile Objekte mit CORBA

Systemtechnik Labor
4BHITY 2015/16, Gruppe X

Hauer Miriam

Version 0.1

Note:

Betreuer: Borko

Begonnen am 29. April 2016

Beendet am 6.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.

2 Quellen

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

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

"JacORB - The free Java implementation of the OMG's CORBA standard."; 03.11.2015; online:

http://www.jacorb.org/

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

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

"Java IDL"; http://docs.oracle.com/javase/7/docs/technotes/guides/idl/jidlExample.html

"expected classname before { token"; http://stackoverflow.com/questions/23507482/c-expected-class-name-before-token http://www.cplusplus.com/forum/general/5973/

Bewertung: 16 Punkte

- ORB-Anpassung in Ordnung und Verwendung korrekt (3 Punkte)

- Mindestens zwei unterschiedliche Programmiersprachen verwendet und IDL-Compiler korrekt angewandt (4 Punkte)

- IIOR über entweder Argument (1 Punkt) / File (3 Punkte) / Namingservice (4 Punkte) verteilt

- Deployment und einfache Tests vorhanden (2 Punkte)

- Protokoll entsprechend der Richtlinien mit entsprechendem theoretischen Background (3 Punkte)

3 Ergebnisse

Alle entstandenen Dateien finden sich im GitRepo :
https://github.com/mhauer-tgm/Syt

3.1 Setups

Ich habe zwei ORB's auf einer Debian VM installiert. Hierfür benötigt man für den Java Part JacORB und für den C++ Part OmniORB. Zuvor habe ich die von Herr Prof. Borko zur Verfügung gestellten code examples in einen neuen Ordner geklont.

```
mkdir repos
git clone URI
```

3.1.1 OmniORB Setup

Es hat sich herausgestellt, dass dieser Teil der komplexere ist. Im folgenden Abschnitt wird der Vorgang dokumentiert.
Zunächst habe ich omniorb mittels wget heruntergeladen und danach in ein eigenes Verzeichnis entpackt.

```
wget https://sourceforge.net/projects/omniorb/files/omniORB/omniORB-4.1.2
tar -xf omniorb
```

Danach erstellen wir ein neues Verzeichnis build indem wir OmniORB builden wollen.

```
mkdir build
../configutr
```

Solange wir keinen C-Compiler installiert haben funktioniert das allerdings nicht also installieren wir ihn nach.

```
apt-get install gcc
```

Danach sollte dieser Schritt problemlos von statten gehen.
Nun habe ich, im sources.list, die Debian Version "jessie" durch "testing" ersetzt und build essentials herunter geladen um die weiteren Schritte möglich zu machen.
Um beim Builden weiterzumachen führen wir den Befehl make im Verzeichnis aus. Diesmal fehlt python, also installieren wir auch dies nach.

```
make
apt-get install libpython 2.7-dev
```

Zum Abschluss des Buildvorgangs führen wir jetzt make install aus, falls hierbei ein "no such file or directory" Fehler auftritt refreshen wir den cash. Danach funktioniert es.

```
make install
ldconfig
```

Schlussendlich muss noch ein Ordner für den Namingservice erstellt werden.Dieser Service kann dann mit folgendem Command gestartet werden :

```
mkdir /var/omninames
omniNames -start -always
```

3.1.2 JacORB Setup

Da wir für JacORB nur die binarys verwenden ist das Setup um einiges einfacher und schneller. Im folgenden Abschnitt wird der Vorgang dokumentiert.
Wie oben haben wir die binarys mittels wget herunter geladen.

http://www.jacorb.org/download.html/jacorb-3.7-binary.zip

Diesmal erstellen wir einen Ordner namens opt und verschieben die entpackten Dateien dort hinein.

mv Downloads/jacorb-3.7 opt

Zum ersten Testen müssen wir noch den path im build.xml und im Server Makefile auf unsere gegebenen Bedingungen anpassen. In meinem Fall :

build.xml : /home/zeljko/repos/opt/jacorb

OMNI_HOME : = /home/zeljko/repos/downloads/omniORB-4.2.1

OMNIIDL : = omniidl

3.2 Test Hallo Welt

Nun wolllen wir das von Herr Prof. Borko zur Verfügung gestelle Test-Programm ausführen. Dazu sind 3 Schritte nötig :

1. omniNames -start -always

```
root@debian:/home/zeljko/repos/code-examples/corba# omniNames -start -always
omniNames: (0) 2016-05-05 20:23:09.950826: Data file: '/var/omninames/omninames-debian.
dat'.
omniNames: (0) 2016-05-05 20:23:09.952947: Read data file '/var/omninames/omninames-deb
ian.dat' successfully.
omniNames: (0) 2016-05-05 20:23:09.953712: Root context is IOR:010000002b00000049444c3a
6f6d672e6f72672f436f734e616d696e672f4e616d696e67436f6e746578744578743a312e3000000100000
0000000007400000001010200100000003139322e3136382e3233302e31333600f90a00000b0000004e616d
6553657276696365000300000000000000080000000100000000545441010000001c0000000100000001000
1000100000001000105090101000100000009010100035454410800000f58b235701006727
omniNames: (0) 2016-05-05 20:23:09.954177: Checkpointing Phase 1: Prepare.
omniNames: (0) 2016-05-05 20:23:09.970270: Checkpointing Phase 2: Commit.
omniNames: (0) 2016-05-05 20:23:09.970706: Checkpointing completed.
```

2. make run im Server Verzeichnis

```
root@debian:/home/zeljko/repos/code-examples/corba/halloWelt/server# make run
# Start Naming service with command 'omniNames -start -always' as root
./server -ORBInitRef NameService=corbaname::localhost
IOR:010000001800000049444c3a68656c6c6f776f726c642f4563686f3a312e3000010000000000
0000680000000101020010000000313932e3136382e3233302e3133360075930000e000000fe43
902b57000003c3000000000000000200000000000000080000000100000000545441010000001c00
0000100000001000100010000000100010509010100010000000901010
```

3. ant run-client

```
root@debian:/home/zeljko/repos/code-examples/corba/halloWelt/client# ant run-cli
ent
Buildfile: /home/zeljko/repos/code-examples/corba/halloWelt/client/build.xml

idl.taskdef:

idl:
   [jacidl] processing idl file: /home/zeljko/repos/code-examples/corba/halloWel
t/idl/echo.idl

compile:

run-client:
     [java] Der Server sagt: Hallo Welt!

BUILD SUCCESSFUL
Total time: 8 seconds
```

3.3 Entwicklung Programm

Im folgenden Abschnitt wird die Entwicklung meines eigenen Programms beschrieben. Als Vorlage dafür verwende ich die jeweils zur Verfügung gestellten Demo-Programme verwendet.

3.3.1 Server Seite

Unser C++ Server benötigt das Programm an sich und das zugehörige Makefile um den Pfad zum IDL Compiler und zum ORB angeben zu können. Als Beispiel habe ich callback gewählt und den sb_server unter omniORB/src/examples/callback entnommen.

Makefile :
Das Makefile unterscheidet sich nicht von dem gegebenen weil es, falls ich alles richtig verstanden habe, keinen Grund gab um es zu ändern.

```
CXX             = /usr/bin/g++
CPPFLAGS        = -g -c
LDFLAGS         = -g
OMNI_HOME       = /home/zeljko/repos/downloads/omniORB/build
#OMNIIDL        = $(OMNI_HOME)/bin/omniidl
OMNIIDL     = omniidl
LIBS            = -lomniORB4 -lomnithread -lomniDynamic4
OBJECTS         = echoSK.o server.o
IDL_DIR     = ../idl
IDL_FILE    = $(IDL_DIR)/echo.idl

all server: $(OBJECTS)
    $(CXX) $(LDFLAGS) -o server server.o echoSK.o $(LIBS)

server.o: server.cc
    $(CXX) $(CPPFLAGS) server.cc -I.

echoSK.o: echoSK.cc echo.hh
    $(CXX) $(CPPFLAGS) echoSK.cc -I.

echoSK.cc: $(IDL_FILE)
    $(OMNIIDL) -bcxx $(IDL_FILE)

run: server
    # Start Naming service with command 'omniNames -start -always' as root
    ./server -ORBInitRef NameService=corbaname::localhost

nameservice:
    omniNames -start -always &

clean clean-up:
    rm -rf *.o
    rm -rf *.hh
    rm -rf *SK.cc
    rm -rf server
```

IDL - File :
Das IDL-File verfügt nun, im Gegensatz zum Gegebenen, nichtmehr über ein Interface Echo, sondern über eines für CallBack und eines für server. Für CallBack wird die void Methode call_back mit einem String als Parameter vorgegeben. Für server werden 3 Methoden, eine one_time die einmal aufgerufen wird, eine register, die mehrmals aufgerufen wird und eine shutdown, die dazu dient den Server herunter zu fahren.

```
#ifndef __ECHO_CALLBACK_IDL__
#define __ECHO_CALLBACK__IDL__
module cb {
    interface CallBack {
        void call_back(in string msg);
    };
    interface server {
        void one_time(in CallBack cb, in string msg);

        void register(in CallBack cb, in string msg, in unsigned short period_scs);

        void shutdown();
    };
};
#endif
```

3.3.2 Client Seite

Unser Java Client benötigt ebenfalls das Programm an sich und das zugehörige build.xml.

Java Programm :
Zusätzlich zum gegebenen Java Client werden die Methoden des Echo Objekts aufgerufen. Wir holen uns ein POA Objekt mittels der resolve_initial_references(namingservice) Methode des ORB Objekts. Von diesem POA Objekt aktiviern wir den POA Manager und erstellen danach ein callback objekt. Zuletzt folgt nicht die call_back Methode die die empfangene Nachricht ausgibt.

```
public class Client {
    public static void main(String[] args)  {
        Echo echo;
        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("Echo", "Object");

            /* Aufloesen der Objektreferenzen  */
            echo = ServerHelper.narrow(rootContext.resolve(name));
            POA rootp = (POA) orb.resolve_initial_references("root poa");
            rootp.thePOAManager().acitave();
            CallBack cbcl = callbackHelper.narrow(rootp.servant_to_reference(new Client()));

            short period = 3;
            echo.one_time(cbcl,"Aufruf Callback 1 mal");
            echo.register(cbcl,"Aufruf Callback x mal",period);
            System.out.println("Der Server sagt: " + echo.echoString("Hallo Welt!"));

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

    public void call_back(String msg){
        System.out.println("Msg :"+msg);
    }
}
```

Build File

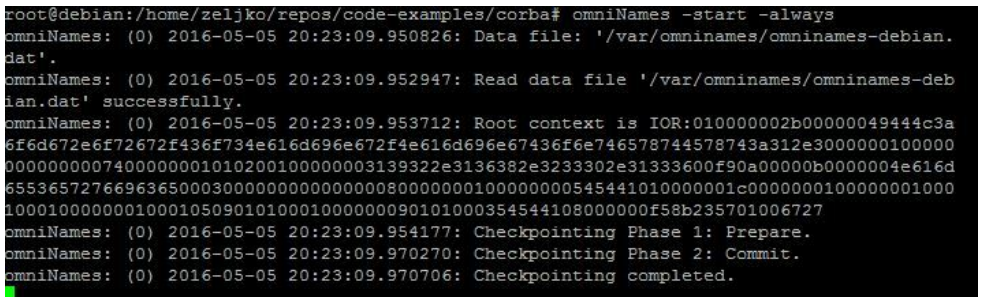
Der Unterschied beläuft sich darauf, dass unter Ausführen des Clients <java fork = “true” classname = “helloworld.Client”> der classname durch cb.Client ersetzt wurde.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xml>

<!--
<project name="client">

    <!-- Setzen aller Variablen -->
    <property name="src.dir" value="src" />
    <property name="build.dir" value="build" />
    <property name="classes.dir" value="${build.dir}/classes" />
    <property name="doc.dir" value="doc" />
    <property name="idl.dir" value="../idl" />
    <property name="gen.dir" value="${build.dir}/generated" />
    <property name="resources.dir" value="resources" />
    <property name="jacorb.dir" value="/root/opt/jacorb" />
    <property name="tmp.dir" value="${build.dir}/tmp" />
    <property name="host" value="127.0.0.1" />

    <!-- Uebergeben der Argumente -->
    <property name="jaco.args" value="-Dignored=value" />

    <!-- Setzen des Classpaths von JacORB -->
    <path id="jacorb.classpath">

        <!-- Setzen des Pfades zu, und inkludieren der Libraries -->
        <fileset dir="${jacorb.dir}/lib">
            <include name="*.jar" />
        </fileset>
    </path>

    <!-- Setzen des Classpaths des Projekts (classes Ordner in build) -->
    <path id="project.classpath">
        <pathelement location="${classes.dir}" />
    </path>

    <!-- Definieren eines in einer bestimmten Klasse vorhandenen Tasks -->
    <target name="idl.taskdef">
        <taskdef name="jacidl" classname="org.jacorb.idl.JacIDL"
            classpathref="jacorb.classpath" />
    </target>

    <!-- Generieren des aus dem idl File resultierenden Quellcodes  -->
    <target name="idl" depends="idl.taskdef">
        <mkdir dir="${idl.dir}" />
        <jacidl srcdir="${idl.dir}" destdir="${gen.dir}" includes="*.idl"
            helpercompat="jacorb" includepath="${jacorb.dir}/idl/omg" />
    </target>

    <!-- Kompilieren des Quellcodes -->
    <target name="compile" depends="idl">
        <mkdir dir="${classes.dir}" />

        <javac destdir="${classes.dir}" debug="true" includeantruntime="false">
            <src path="${gen.dir}" />
            <src path="${src.dir}" />
            <classpath refid="jacorb.classpath" />
        </javac>
    </target>

    <!-- Ausfuehren des Clients -->
    <target name="run-client" depends="compile">
        <description>
            Dem Client kann eine Hostadresse mitgegeben werden.
            Ein Aufruf ist mit 'ant run-client -Dhost=host' moglich.
            Beispielaufruf: ant run-client -Dhost=127.0.0.1

            Sollte kein Host angegeben werden, so wird localhost als Host verwendet.
        </description>
        <java fork="true" classname="cb.Client">

            <!-- Wuerde folgendem Aufruf entsprechen: java helloworld.Client -ORBIni
            <arg value="-ORBInitRef" />
            <arg value="NameService=corbaloc::${host}:2809/NameService" />
            <classpath refid="project.classpath" />
        </java>
    </target>

    <!-- Loeschen des build Ordners -->
    <target name="clean">
        <delete dir="${build.dir}" />
    </target>

</project>
```

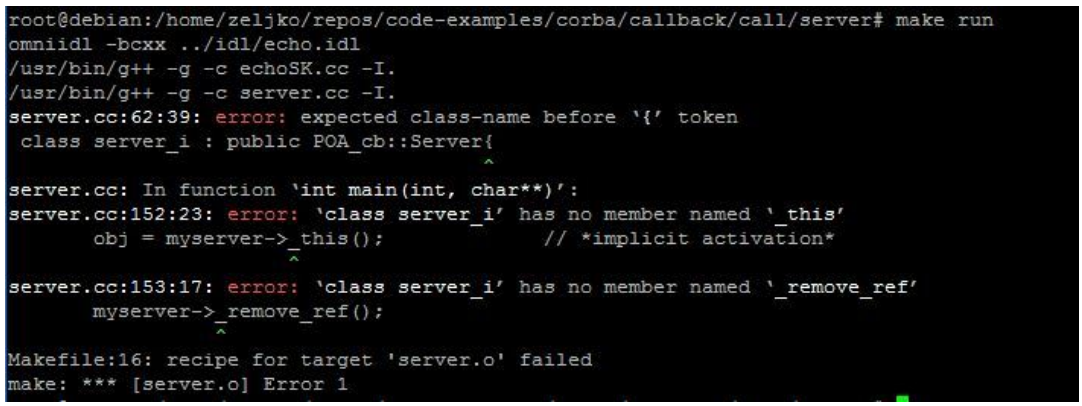
3.3.3 Testen des Programms

Um das eigene Programm auszuführen sind die selben Schritte notwendig wie zum Ausführen des Test Hallo Welt Programms.

1. omniNames -start -always



2. make run im Server Verzeichnis



3. ant run-client

Ohne den laufenden Server funktioniert natürlich auch der Client nicht.

4 Zeit und Probleme

4.1 Probleme

Abgesehen von der sehr ungewohnten Situation auf der Konsole zu Programmieren weil meine VM über keine GUI verfügte bin ich, trotz intensiver Recherche, über die oben gezeigte Fehlermeldungen nicht hinweggekommen. Vermutlich lässt sich dieser Umstand auf eine fehlerhafte Zeitplanung meinerseits, aufgrund einer unerwarteten Erkrankung, die eine massive Einschränkung mit sich brachte, zurückführen. Auch wenn ich dadurch die Aufgabenstellung nicht erfüllen konnte, lerne ich doch solche Verzögerungen bei meiner persönlichen Planung nicht außer acht zu lassen.

4.2 Zeitaufwände und Einschätzung

Zeitangaben	Schätzung	Aufwand
Stunden	4	5 unfertig