JUST DO IT
HÖHERE ABTEILUNG FÜR INFORMATIONSTECHNOLOGIE

tgm
Die Schule der Technik

Laborprotokoll

DEZSYS-07
VERTEILTE OBJEKTE MIT CORBA

Systemtechnik Labor
4CHIT 2015/16, Gruppe C

Steinwender Jan-Philipp

Version 1.1

Note:

Betreuer: Borko

Begonnen am 08. April 2016

Beendet am 15. April 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 Github repository

Das Projekt steht online auf Github und ist unter diesem Link erreichbar.

https://github.com/jsteinwender-tgm/DEZSYS-07-VERTEILTE-OBJEKTE-MIT-CORBA.git

1.5 Quellen aus/in der Angabe

"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

2 Corba

2.1 Einleitung

„CORBA (Common Object Request Broker Architecture) unterstützt verteilte Anwendungen, die plattformübergreifend also unabhängig von Sprache und Betriebssystem sind. Darin unterscheidet es sich von RMI, das nur zwischen JAVA-Anwendungen möglich ist. CORBA beschreibt die Architektur und ein ORB (Object Request Broker) stellt eine spezielle Implementierung dar.

Ein Server könnte also auf einem Linux-System laufen und in C++ geschrieben sein während ein Client auf einem Windows-PC gestartet wird und in Java geschrieben ist. Der Aufruf selbst ist transparent, d.h. der Nutzer merkt nichts davon, dass die Methode auf einem entfernten Rechner implementiert ist und dort abgearbeitet wird. Parameter und Ergebnis werden transparent zwischen Client und Server ausgetauscht.

CORBA benutzt die IDL (Interface Definiton Language) zur Beschreibung der Schnittstelle zwischen Server und Client, zu der die zu benutzenden entfernten Methoden gehören. Ein IDL-Compiler (für Java 'idlj') generiert daraus die erforderlichen Klassen (stub, skeleton und weitere Hilfsklassen). Im Hintergrund übernehmen diese generierten Klassen stellvertretend für Server/Client Kommunikation und Datenaustausch.“[WBC01]

2.2 Architektur

POA Portable Object Adapter

„Der POA als Teil des ORB vermittelt die Anforderung des Clients (Aufruf der entfernten Methode) an den Server, der diese Methode implementiert hat.

- Vererbungs-Modell (POA):
  Die Implementierungsklasse erbt (per 'extends') von der Skeleton-Klasse, die der IDL-Compiler generiert hat.
- Delegation-Modell (POA/Tie) benutzt 2 Klassen:
  Eine Tie-Klasse erbt vom Skeleton (POA), delegiert aber die Anforderung an die eigentliche Implementierungs-Klasse.

„[WBC01]

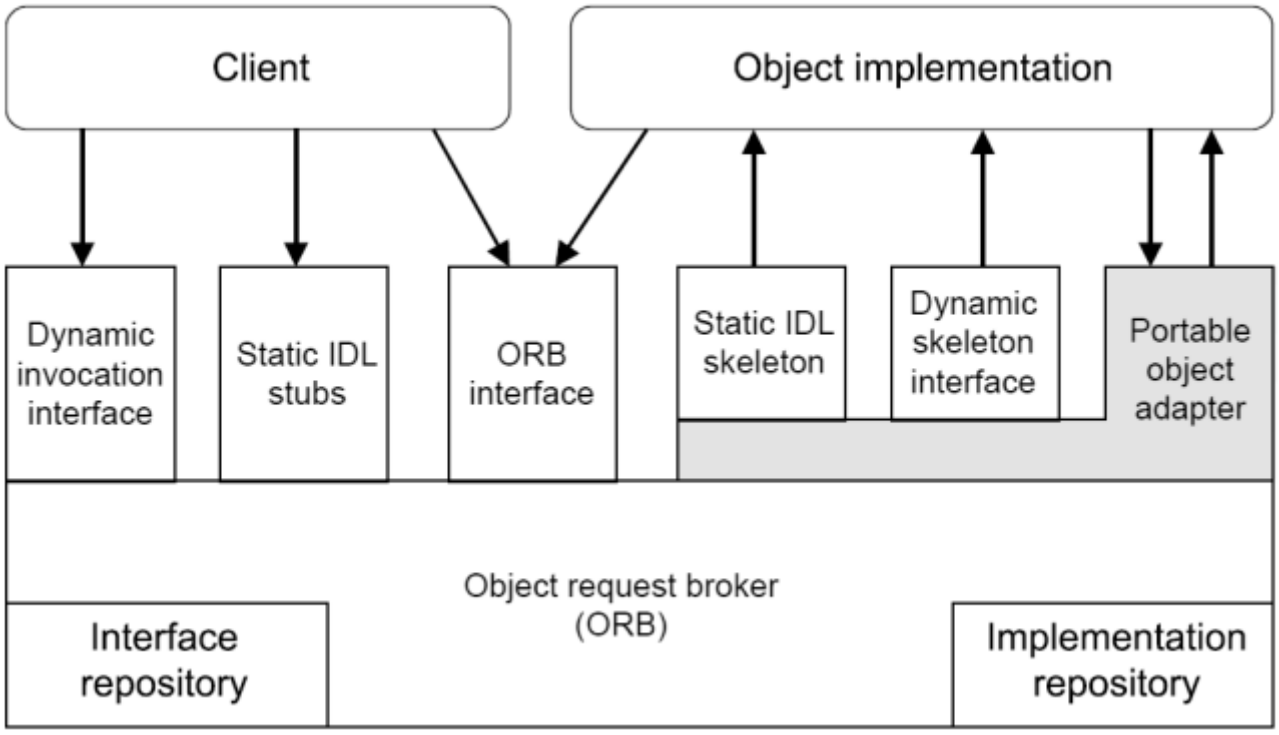

Figure 1Portable Object Adapter[IMG01]

Object Request Broker (ORB)

„Dem ORB liegt folgendes Konzept zugrunde: Wenn eine Anwendungskomponente einen Dienst nutzen möchte, der von einer anderen Komponente zur Verfügung gestellt wird, muss sie zunächst eine Objektreferenz für das Objekt erlangen, welches den Dienst zur Verfügung stellt. Nachdem die Objektreferenz erlangt wurde, kann die Komponente Methoden des Objekts aufrufen und damit auf die vom Objekt zur Verfügung gestellten Dienste zugreifen. Der Entwickler der Client-Komponente weiß zum Zeitpunkt des Kompilierens, welche Methoden eines bestimmten Server-Objekts zur Verfügung stehen. Die Hauptaufgabe des ORB ist die Auflösung der Anfragen von Objektreferenzen, so dass die Anwendungskomponenten die Konnektivität miteinander herstellen können.“[RVA01]

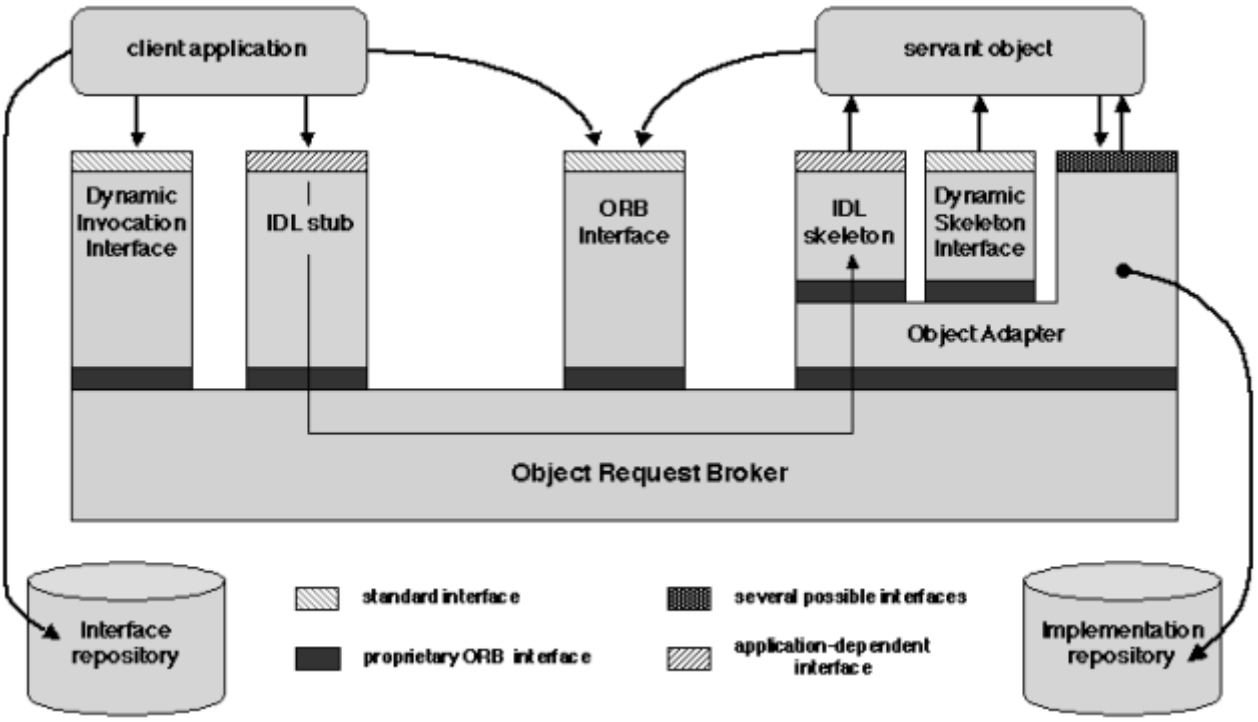

Figure 2Middleware Architecture [IMG02]

„Die Formatübertragung (Marshaling) ist die Umsetzung der Eingabeparameter in ein Format, das über ein Netzwerk zum Remote-Objekt übertragen werden kann. Der ORB macht auch das Umgekehrte (Unmarshaling). Die gesamte Formatübertragung erfolgt ohne jeglichen Eingriff des Programmierers, der Client ruft einfach die gewünschte Remote-Methode auf und bekommt ein Ergebnis zurück, als wäre es eine lokale Methode. Ein anderes Ergebnis von Marshaling besteht darin, dass die Kommunikation zwischen den Komponenten plattformunabhängig stattfindet, weil die Parameter für die Übertragung in ein plattformunabhängiges Format konvertiert werden."[RVA01]

Kommunikationsmodell

„Die CORBA-Spezifikation ist hinsichtlich der Netzwerkprotokolle neutral. Der CORBA-Standard legt das GIOP (General Inter-ORB Protokoll, allgemeines Protokoll für die Kommunikation zwischen ORBs) fest, das auf einer hohen Ebene einen Standard für die Kommunikation zwischen verschiedenen CORBA-ORBs und -Komponenten definiert."[RVA01]

Interface Definition Language (IDL)

„Die IDL dient zur Definition von Schnittstellen zwischen einzelnen Anwendungskomponenten. Sie ist nicht eine prozedurorientierte Sprache, sondern dient einzig und allein zur Definition von Schnittstellen und nicht von Implementierungen (ähnlich einer C++ header-Datei). Die IDL-Spezifikation ist dafür verantwortlich, dass die Daten zwischen Anwendungen in unterschiedlichen Sprachen korrekt ausgetauscht werden. Wenn es sich z.B. beim IDL-Typ long um eine vorzeichenbehaftete 32 Bit große Zahl handelt, kann diese ein C++ long sein (je nach Plattform) oder ein Java int.“[RVA01]

„IDL verwendet ORB spezifische IDL Compiler zur Generierung von Stub und/oder Skeleton Code. Die Syntax von IDL ist an der von C++ angelehnt, wobei sich aber die Schlüsselwörter unterscheiden.“[PDC01]

3 Vorbereitung

Zuerst muss omniORB und Jacob heruntergeladen, dann entpackt und konfiguriert werden. (binary, direct download link)

omniOrb:

```
~/Download$ wget
http://downloads.sourceforge.net/project/omniorb/omniORB/omniORB-
4.2.1/omniORB-4.2.1-
2.tar.bz2?r=https%3A%2F%2Fsourceforge.net%2Fprojects%2Fomniorb%2Ffiles%2F
omniORB%2FomniORB-4.2.1%2F&ts=1460116058&use_mirror=liquidtelecom
~/Download$ mv omniORB-4.2.1-2.tar.bz2
~/Download$ bunzip2 omniORB-4.2.1-2.tar.bz2
~/Download$ tar –xf omniORB-4.2.1-2.tar
```

Jacob:

```
~/Download$ wget http://www.jacorb.org/releases/3.7/jacorb-3.7-binary.zip
~/Download$ unzip jacob
```

Ein Verzeichnis „`~/opt`“ anlegen in das wir die beiden entpackten Ordner verschieben. (Tipp: Ordner umbenenne, ohne Version etc.) Es ist zu empfehlen die `ReadMe`(vor allem von omniORB) zu lesen da hier auch die Konfiguration beschrieben ist. Das wichtigste ist:

```
cd $OMNIORB_TOP
$ mkdir build
$ cd build
```

```
$ ../configure [configure options]// geht noch nicht
$ make
$ make install        //setzt die path variable
```

Es werden auch noch weitere Pakete benötigt die möglicher Weise nicht installiert sind.

```
# apt-get update

# apt-get install gcc make autoconf automake gcc-multilib pkg-config g++
libpython2.7-dev ant openjdk-8-jdk
```

Um später den omniNames Service zu starten müssen wir jetzt noch eine Umgebungsvariable definieren.

```
~# nano .bashrc
      ……
      if [ -d $HOME/opt/omniORB/lib ]; then
            export LD_LIBRARY_PATH="/home/jstone/opt/omniORB/lib"
      fi
```

4 Bsp#1 HelloWorld

Das erste Beispiel ist ein simples HelloWorld. Den Beispiel-Code kann man von Github laden. Man findet es unter „`code-examples-master/corba/halloWelt/`“.

4.1 Namesservice

Für den Service müssen wir vorerst nichts verändern. Jedoch sollten wir gleich einen Ordner anlegen.: Auf Kleinschreibung Achten!

```
# mkdir /var/omninames/
```

Hat man das Verzeichnis erstellt kann man den Server auch schon starten.:

```
# omniNames –start –always      //eventuell auch mal den server starten
```

4.2 Server

Im „`Makefile`“ muss man die Datei an sein eigenes System anpassen. Die Änderungen sind makiert, den Rest behalten wir bei.:

```
# nano Makefile
      CXX             = /usr/bin/g++
      CPPFLAGS        = -g –c
      LDFLAGS         = -g
```

```
OMNI_HOME        = /home/mike/opt/omniORB
OMNIIDL          = $(OMNI_HOME)/ build/bin/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)

……
```

Haben wir das „Server-Konfig-File" angepasst können wir den Server auch schon mal zu builden anfangen.

```
# make
# make server
# make run       //könnte noch nicht funktionieren
```

4.3 Client

Der Client wird mit „ant" gestartet. Die Konfiguration dafür ist eine „`build.xml`" im Ordner „`…/helloworld/client`". Auch die müssen wir anpassen:

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

     <!--
          ANT Build File des Hallo Welt CORBA Clients.

          Author: Hagen Aad Fock hfock@student.tgm.ac.at
          Version: 1.0
          Datum: 22.02.2015
     -->

     <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="/home/jstone/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" />
```

Hat man das nun auch erledigt, kann man nun auch den Client builden.:

```
ant clean
ant compile
ant run-client
```

Hat alles funktioniert sollte beim Client-Terminal eine Ausgabe wie unten kommen.:

```
jstone@debian:~/code-examples-master/corba/halloWelt/client$ ant run-client
Buildfile: /home/jstone/code-examples-master/corba/halloWelt/client/build.xml

idl.taskdef:

idl:
   [jacidl] processing idl file: /home/jstone/code-examples-master/corba/halloWelt/idl/echo.idl

compile:

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

BUILD SUCCESSFUL
Total time: 1 second
jstone@debian:~/code-examples-master/corba/halloWelt/client$
```

Figure 3Ausgabe bei Erfolg

5 Bsp#2 Callback

Dieses Beispiel verläuft ähnlich wie das vorherige HelloWorld.
Es wird wieder ein Beispiel-Code verwendet, dieser liegt „…/omniORB/src/examples/call_back/". Den Ordner „call_back" kopieren wir uns in unser vorheriges Verzeichnis, „…/corba/".

5.1 Vorbereitung

In dem Verzeichnis(„call_back") erstellen wir drei neue Ordner, client, idl und server.

```
$ mkdir client idl server
```

5.2 Nameservice

Man verschiebt die Datei „echo_callback.idl" in das Verzeichnis „idl".

```
$ mv echo_callback.idl idl/
# omniNames -start -always      //als root
```

Nichts in dem File ändern!

5.3 Server

Wir kopieren uns das Makefile von der helloWorld Aufgabe. Hier muss man einiges anpassen.:

```
$ mv cb_server.cc server/
$ cp ../halloWelt/server/Makefile server/
$ nano server/Makefile
     CXX             = /usr/bin/g++
     CPPFLAGS        = -g -c
     LDFLAGS         = -g
     OMNI_HOME       = /home/jstone/opt/omniORB
     OMNIIDL         = $(OMNI_HOME)/build/bin/omniidl
     LIBS            = -lomniORB4 -lomnithread -lomniDynamic4
     OBJECTS         = echo_callbackSK.o cb_server.o
     IDL_DIR         = ../idl
     IDL_FILE        = $(IDL_DIR)/echo_callback.idl
     LD_LIBRARY_PATH = $(OMNI_HOME)/lib

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

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

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

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

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

clean clean-up:
     rm -rf *.o
```

```
rm -rf *.hh
rm -rf *SK.cc
rm -rf cb_server
```

Nun wie gehabt wieder den Server builden und in Betrieb nehmen.:

```
# make
# make server
# make run
```

5.4 Client

Den Client wollen wir wieder mithilfe von ant builden und ausführen, deshalb kopieren wir uns ebenfalls die build.xml von der Aufgabe helloWorld und verändern diese.:

```
$ cp ../halloWelt/client/build.xml client/
$ cp …/Client.java client/
$ nano client/Client.java
      <?xml version="1.0" encoding="UTF-8"?>
      <!DOCTYPE xml>

      <!--
            ANT Build File des Callback CORBA Clients.
            Author: Hagen Aad Fock hfock@student.tgm.ac.at
            Version: 1.0
            Datum: 22.02.2015
      -->

      <project name="client">

      ……
            <target name="run-client" depends="compile">
                  <description>
                        Dem Client kann eine Hostadresse mitgegeben werden.
                        Ein Aufruf ist mit 'ant run-client -Dhost=host'
                        möglich. 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
                        call_back.Client -ORBInitRef
                        NameService=corbaloc::127.0.0.1:2809/NameService
                        -->
                        <arg value="-ORBInitRef" />
```

```
                    <arg value="NameService=corbaloc:
                          :${host}:2809/NameService" />
                    <classpath refid="project.classpath" />
                </java>
            </target>

        .....
```

```
root@debian:/home/jstone/code-examples-master/corba/call_back/server# make run
# Start Naming service with command 'omniNames -start -always' as root
./cb_server -ORBInitRef NameService=corbaname::localhost
IOR:010000001200000049444c3a63622f5365727665723a312e3000000001000000000000006800000000
8342e31333100008e8700000e000000fe1f4e0c5700000658000000000000000200000000000000080000
0001000000010001000100000000100010509010100010000000090101000
cb_server: Doing a single call-back: Server: Heylo
cb_server: Starting a new worker thread
```

Figure 4 Ausgabe des Servers wenn wir ihn starten.

```
jstone@debian:~/code-examples-master/corba/call_back/client$ ant run-client
Buildfile: /home/jstone/code-examples-master/corba/call_back/client/build.xml

idl.taskdef:

idl:
   [jacidl] processing idl file: /home/jstone/code-examples-master/corba/call_

compile:

run-client:
     [java] Client Callback: Nachricht erhalten: Server: Heylo
     [java] Client Callback: Nachricht erhalten: Client: How you doing?
     [java] Client Callback: Nachricht erhalten: Client: How you doing?

     [java] Verbindung getrennt!

BUILD SUCCESSFUL
Total time: 3 seconds
jstone@debian:~/code-examples-master/corba/call_back/client$
```

Figure 5 Ausgabe des Client nach dem er gestartet ist und der Server auch zur Verfügung steht.

```
cb_server: Lost a client!
cb_server: Worker thread is exiting.
```

Figure 6Ausgabe des Server nachdem die Verbindung beendet wurde.

6 Zeitschätzung

Fertigstellung	Beschreibung	Geschätzte Zeit in min	Benötigte Zeit in min
14.04.2016	omniORB zum Laufen bringen	240	360
14.04.2016	Demo Programm HelloWorld	120	240
15.04.2016	Demo Programm Callback	120	210
15.04.2016	Protokoll	90	120
	Insgesamt	**570**	**930**

7 Quellen

Bsp#1 https://github.com/mborko/code-examples

[WBC01] Java Standard: Corba. Wikibooks [ONLINE]. AVAILABLE AT: https://de.wikibooks.org/wiki/Java_Standard:_Corba#Einleitung.2C_Grundbegriffe [ABGERUFEN AM 15.04.2016]

[PDC01] Überblick über OMG Interface Definition Language. Projektstudium SS98 - distributed computing. André Möller, Oliver Mentz & Magnus Wiencke [ONLINE]. AVAILABLE AT: http://www.fh-wedel.de/~si/projekte/ss98/Ausarbeitung/DistributedProgramming/Ueberblick_ueber_OMG_Interface_Definition_Language.html [ABGERUFEN AM 15.04.2016]

[PDC02] Überblick über Java ORBs. Projektstudium SS98 - distributed computing. André Möller, Oliver Mentz & Magnus Wiencke [ONLINE]. AVAILABLE AT: http://www.fh-wedel.de/~si/projekte/ss98/Ausarbeitung/DistributedProgramming/Ueberblick_ueber_Java_ORBs.html [ABGERUFEN AM 15.04.2016]

[RVA01] CORBA (Überblick, IDL). Friedrich-Alexander-Universität Erlangen-Nürnberg. Radu Vatav. [ONLINE]. AVAILABLE AT: https://www4.informatik.uni-erlangen.de/DE/Lehre/WS04/PS_KVBK/talks/CORBA-ausarbeitung.pdf [ABGERUFEN AM 15.04.2016]

[IMG01] Distributed Systems Technologies, Lorenz Froihofer.

[IMG02] „Middleware Architecture with Patterns and Frameworks". S.Krakowiak. [OFFLINE]. WAS AVAILABLE AT: http://sardes.inrialpes.fr/~krakowia/MW-Book/Chapters/DistObj/distobj-body.html [ZULETZT ONLINE 2012]