

hallo corba!

SYT / DEZSYS08



20. März 2015

erceg. Geyer

4AHIT

Inhalt

[1. Aufgabenstellung 2](#_Toc417378501)

[2. Requirementanalyse & Aufwandabschätzung 2](#_Toc417378502)

[3. anschließende Endzeitaufteilung 3](#_Toc417378503)

[3.1 Erceg 3](#_Toc417378504)

[3.2 Geyer 3](#_Toc417378505)

[3.3 Gesamtsumme 3](#_Toc417378506)

[4. Arbeitsdurchführung 4](#_Toc417378507)

[4.1 OmniORB installieren und konfigurieren 4](#_Toc417378508)

[4.2 OmniORB Examples ausführen 4](#_Toc417378509)

[4.3 IDL Files in OmniORB umwandeln 5](#_Toc417378510)

[4.4 IDL Files in Java umwandeln 6](#_Toc417378511)

[4.5 OmniORB Naming Service 6](#_Toc417378512)

[4.6 Java CORBA Programmargumente 6](#_Toc417378513)

[5. Lessons learned 7](#_Toc417378514)

[6. Quellenangaben 7](#_Toc417378515)

*Github-Link:* [*https://github.com/serceg-tgm/DezSys08*](https://github.com/serceg-tgm/DezSys08)

*Github-Tag: erceg\_geyer\_dezsys08\_v1*

# 1. 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 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.

**Resources**

<http://omniorb.sourceforge.net/>   
<http://www.microfocus.com/products/corba/orbacus/>   
<http://www.jacorb.org/>   
<http://omniorb.sourceforge.net/omni41/omniORB.pdf>   
<http://www.ing.iac.es/~docs/external/corba/book.pdf>

# 2. Requirementanalyse & Aufwandabschätzung

|  |  |  |  |
| --- | --- | --- | --- |
| Arbeitspaket | beteiligte Teammitglieder | Geschätzte Zeit | Erledigt |
| omniORB auf der virtuellen Maschine zum Laufen bringen | Erceg, Geyer | 2 h | x |
| Java Implementierung des Objekts | Erceg, Geyer | 3 h | x |
| C++ Implementierung des Objekts | Erceg, Geyer | 2 h | x |
| IDL - Interface | Erceg, Geyer | 1 h | x |
| Namingservice in Verwendung | Erceg, Geyer | 10 h | x |
| Ausgeführte Schritte im Protokoll erläutern | Erceg, Geyer | 4 h | x |
|  |  |  |  |
| *Gesamt* |  | **22 h** |  |

# 3. anschließende Endzeitaufteilung

## 3.1 Erceg

|  |  |  |
| --- | --- | --- |
| Arbeit | Datum | Zeit in Minuten |
| omniORB zum Laufen gebracht | 16.01.2015 | 100 Minuten |
| Protokoll geschrieben | 16.01.2015 | 50 Minuten |
| C++ Implementierung in das Projekt eingebunden | 21.01.2015 | 50 Minuten |
| Java Implementierung begonnen | 05.02.2015 | 80 Minuten |
| Java Example erklärt | 14.02.2015 | 100 Minuten |
| Java Implementierung weitergeführt | 14.02.2015 | 100 Minuten |
| Protokoll geschrieben | 14.02.2015 | 30 Minuten |
| Protokoll fertiggestellt | 20.03.2015 | 30 Minuten |
|  |  |  |
| *Gesamt* | *20.03.2015* | **540 Minuten (9 h)** |

## 3.2 Geyer

|  |  |  |
| --- | --- | --- |
| Arbeit | Datum | Zeit in Minuten |
| omniORB zum Laufen gebracht | 16.01.2015 | 100 Minuten |
| Protokoll geschrieben | 16.01.2015 | 50 Minuten |
| C++ Implementierung in das Projekt eingebunden | 21.01.2015 | 50 Minuten |
| IDL – Interface beschrieben | 14.02.2015 | 100 Minuten |
| Java Implementierung fertiggestellt | 14.02.2015 | 120 Minuten |
| Java Example erklärt | 14.02.2015 | 100 Minuten |
| Protokoll geschrieben | 14.02.2015 | 30 Minuten |
| Namingservice in Verwendung gebracht | 20.03.2015 | 90 Minuten |
|  |  |  |
| *Gesamt* | *20.03.2015* | **640 Minuten (10 h 40 min)** |

## 3.3 Gesamtsumme

Es wurde geschätzt, dass wir für diese Übung 22 Stunden benötigen. Mit unserer Gesamtzeit von **19 Stunden und 40 Minuten** wurde bewiesen, dass unsere Einschätzung ziemlich gut geplant war. Vermutlich wäre die Aufgabe auch früher fertig geworden, hätten wir nicht Probleme mit dem Namingservice gehabt und somit Herrn Professor Borko um Hilfe bitten mussten (danke an dieser Stelle noch einmal!).

# 4. Arbeitsdurchführung

## 4.1 OmniORB installieren und konfigurieren

1. **OmniORB von Sourceforge downloaden**

wget http://sourceforge.net/projects/omniorb/files/omniORB/omniORB-4.2.0/omniORB-4.2.0.tar.bz2/download

1. **OmniORB konfigurieren**

Durch Ausführen des Configure-Scripts (./configure) werden die Anforderungen überprüft und notwendige Variablen gesetzt.

In der Datei /etc/omniORB.cfg muss der Wert

InitRef = NameService=corbaname::my.host.name

auskommentiert und wie folgt ersetzt werden:

InitRef = NameService=corbaname::127.0.0.1

## 4.2 OmniORB Examples ausführen

Die folgende Dokumentation betrifft Linux Systeme (getestet auf Debian x64, OmniORB 4.2.0).

1. **Examples kompilieren**

Zum Testen wurde das echo Example benutzt. (src/examples/echo)

Die Files wurden mit make all kompiliert.

1. **Examples ausführen**

Zum Ausführen der Examples müssen Libraries (libomniORB4.so) geladen werden, auf welche das Programm derzeit nicht zugreifen kann. Das kann durch eine globale Variable oder einen Eintrag im Path behoben werden.

LD\_LIBRARY\_PATH=/home/schueler/omniorb/omniORB-4.2.0/lib

export LD\_LIBRARY\_PATH

Diese Variable sollte zur .bashrc hinzugefügt werden. Anderenfalls funktioniert nur für eine Session und muss nach jedem Neustart oder Ähnlichem neu gesetzt werden.

Nun können die Beispiele ausgeführt werden.

Das erste Beispiel (./eg1) beinhaltet eine Applikation den Server und den Client.

Beim Ausführen sendet der Client das IDL Objekt an den Server, welcher es wieder in ein C++ Objekt umwandelt und dieses wiederum ausgibt.

Beim zweiten Beispiel sind diese Aufgaben aufgeteilt.

Einerseits gibt es das Programm eg2\_impl, welches nach Aufruf (ohne Parameter) eine IOR (Interoperable Object Reference, eine Referenz auf ein CORBA Objekt [1]) ausgibt. Da das Programm den Server darstellt, läuft es nach der Ausgabe weiter und wartet auf eine Nachricht vom Client.

Die generierte IOR kann z.B. so aussehen:

$ ./eg2\_impl

IOR:010000000d00000049444c3a4563686f3a312e300000000001000000000000006800000001010200100000003139322e3136382e3131382e31333100c6c800000e000000fe6789d3540000272f000000000000000200000000000000080000000100000000545441010000001c00000001000000010001000100000001000105090101000100000009010100

Um das Programm eg2\_clt zu starten, muss die vom Server generierte IOR als Argument angegeben werden:

$ ./eg2\_clt IOR:010000000d00000049444c3a4563686f3a312e300000000001000000000000006800000001010200100000003139322e3136382e3131382e31333100c6c800000e000000fe6789d3540000272f000000000000000200000000000000080000000100000000545441010000001c00000001000000010001000100000001000105090101000100000009010100

Nach dem Aufruf erfolgt ein Austausch zwischen Server und Client und die Nachricht wird gesendet.

Ausgabe Server:

Upcall: Hello!

Ausgabe Client:

I said, "Hello!".

The Echo object replied, "Hello!".

Um Beispiel 3 auszuführen wird ein COS Naming Service benötigt. Weiteres dazu wird in Punkt 5.5 behandelt.

## 4.3 IDL Files in OmniORB umwandeln

Durch das mit OmniORB installierte Programm omniidl kann aus einem IDL File ein Header und ein Stub für eine Implementierung generiert werden:

omniidl -bcxx echo.idl

Für das benutzte Beispiel wurde das folgende IDL File erstellt (generiert von OmniORB)

#ifndef \_\_ECHO\_IDL\_\_

#define \_\_ECHO\_IDL\_\_

interface Echo {

string echoString(in string mesg);

};

#endif // \_\_ECHO\_IDL\_\_

## 4.4 IDL Files in Java umwandeln

Wie für OmniORB muss auch für Java das IDL File kompiliert werden sodass es verwendet werden kann. Java hat das Programm idlj mitgeliefert (zu finden unter C:\Programme\Java\jdk1.8.0\_20\bin\idlj.exe).

Das File muss über den folgenden Befehl kompiliert werden (falls Java im PATH eingetragen ist):

idlj –fall –td <target-dir> <file-location>

Also z.B.

idlj -fall -td C:\Users\Stefan\Desktop\hello C:\Users\Stefan\Desktop\hello\Hello.idl

Falls das Programm richtig angewandt wurde, sollte ein Ordner generiert worden sein, welcher alle für das zu übertragende Objekt notwendigen Informationen enthält. Diese können nun in das Projekt eingebunden werden. [2] [3]

## 4.5 OmniORB Naming Service

Das Naming Service erstellt Files und kann nur gestartet werden, wenn diese Files nicht existieren. Daher ist es Vorteilhalt dich einen Datenfolder für diese Files anzulegen. Das Naming Service wird wie folgt gestartet.

omniNames -start 1050 -datadir names -ORBsupportBootstrapAgent 1

omniNames –start <port> -datadir <path-of-data-files> -ORBsupportBootstrapAgent 1

Damit kann nun auch das 3. Testprogramm gestartet werden.

Das Naming Service kann mit dem folgenden Kommando gestoppt werden:

## 4.6 Java CORBA Programmargumente

Um den ORB zu initialisieren muss ein Argument mit den Daten des Servers angegeben werden.

Dieses lautet wie folgt:

-ORBInitRef NameService=corbaloc::vmdebian:1050

oder

-ORBInitialPort 1050 -ORBInitialHost vmdebian

# 5. Lessons learned

Wir haben gelernt…

* einen OmniORB Server zu konfigurieren
* die Client Konfigurationen anzupassen und eine Nachricht an den Server zu schicken

# 6. Quellenangaben

[1] Wikipedia (2013). Interoperable Object Reference [Online].

Available at: <http://de.wikipedia.org/wiki/Interoperable_Object_Reference>

[zuletzt abgerufen am: 05.02.2014]

[2] Oracle (1993, 2014). Java IDL: The „Hello World“ Example [Online]. Available at:

<http://docs.oracle.com/javase/7/docs/technotes/guides/idl/jidlExample.html>

[zuletzt abgerufen am: 14.02.2014]

[3] Oracle (1993, 2014). idlj – The IDL-to-Java Compiler [Online]. Available at:

<http://docs.oracle.com/javase/7/docs/technotes/tools/share/idlj.html>

[zuletzt abgerufen am: 14.02.2014]