

**Laborprotokoll**

**DezSys-07 “Verteilte Objekte mit Corba”**

**Systemtechnik Labor**

**4CHIT 2015/16, Gruppe A**

**Gabriel Frassl**

|  |  |
| --- | --- |
|  | **Version 1.0** |
| **Note:** | **Begonnen am 15. April 2016** |
| **Betreuer: M.Borko** | **Beendet am 22. April 2016** |

Inhalt

[1 Einführung 3](#_Toc449091217)

[1.1 Ziele 3](#_Toc449091218)

[1.2 Voraussetzungen 3](#_Toc449091219)

[1.3 Aufgabenstellung 3](#_Toc449091220)

[2 Ergebnisse 4](#_Toc449091221)

[2.1 CORBA Theorie [1] 4](#_Toc449091222)

[Einführung 4](#_Toc449091223)

[Aufbau 4](#_Toc449091224)

[2.2 CORBA Setup 7](#_Toc449091225)

[OmniOrb Installation/Setup 7](#_Toc449091226)

[JacORB Installation/Setup 8](#_Toc449091227)

[Ordner /opt 8](#_Toc449091228)

[Zusätzliche Pakete 8](#_Toc449091229)

[2.3 Callback Anwendung 9](#_Toc449091230)

[Server Seite 9](#_Toc449091231)

[IDL File 10](#_Toc449091232)

[Client Seite 11](#_Toc449091233)

[2.4 Testen 13](#_Toc449091234)

[HelloWorld testen 14](#_Toc449091235)

[Callback testen 14](#_Toc449091236)

[3 Quellen / Literatur 15](#_Toc449091237)

[4 Zeitaufwand/ Probleme 15](#_Toc449091238)

[5 GITHUB 16](#_Toc449091239)

# 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.

## 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.

## Voraussetzungen

* Grundlagen Java, C++ oder anderen objektorientierten Programmiersprachen
* Grundlagen zu verteilten Systemen und Netzwerkverbindungen
* Grundlegendes Verständnis von nebenläufigen Prozessen

## 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.

# Ergebnisse

## CORBA Theorie [1]

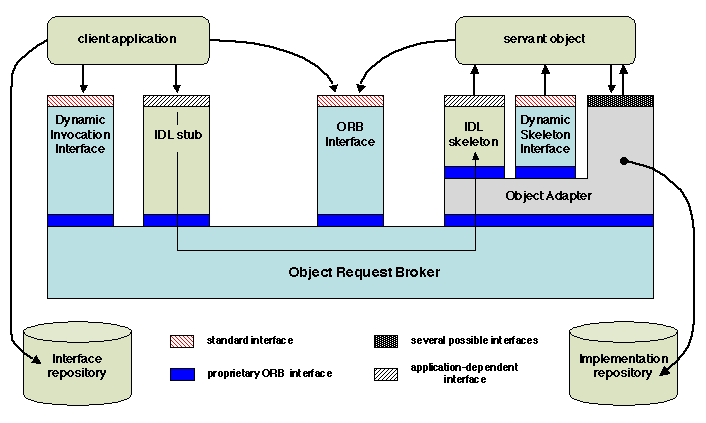
### Einführung

CORBA (COMMON OBJEKT REQUEST BROKER ARCHITECTURE) ist eine Architektur die für das Erstellen von verteilten Anwendungen, und dem Aufruf eben dieser dient.

Wie das Wort COMMON im Namen schon sagt ist CORBA eine plattformübergreifende Lösung. Das heißt, dass damit eine Anwendung erstellt werden kann die mit mehreren Programmiersprachen funktioniert. So kann zum Beispiel ein Java Programm die Methode eines verteilten C++ Programmes aufrufen.

Dabei liefert CORBA selbst nur die Spezifikationen aufgrund derer andere Unternehmen oder Communitys dann eine eigene Implementation in Form eines ORB’s für eine bestimmte Sprache erstellen. Aufgrund der gemeinsamen CORBA Schnittstelle können diese ORB’s miteinander kommunizieren.

### Aufbau



**Object Request Broker:**

Wie bereits erwähnt sind die ORB’s Implementierungen des von CORBA spezifizierten ORB Interfaces.

Sowohl Client als auch der Server haben Zugriff auf diesen ORB da dieser für jegliche Kommunikation zwischen den beiden zuständig ist. Die Funktionsweise eines ORB ähnelt dem eines Datenbus.

Sämtliche Services wie zum Beispiel der Naming service werden ebenfalls vom ORB gesteuert und angesprochen.

**(Portable) Objekt Adapter (P)OA**

Der POA hat die Aufgabe alle benötigten Objekte und Interfaces auf Serverseite zu verwalten, und bei einem Aufruf eines Clients für die Bereitstellung des Objektes und möglicherweise des Skeletons zu sorgen.

* Generieren der Objekt Referenz
* Methodenaufruf
* Security
* Mappen von Objekt Referenz mit Objekt Implementierung
* Implementierung registrieren

**Interface Definition Language / IDL Compiler**

Die IDL ist ein kritischer Bestandteil der Plattformunabhängigkeit von CORBA, da bei der Method Invokation sowohl auf Clientseite als auch auf Serverseite Interfaces (Stub, Skeleton) vorhanden sein müssen, die den Zugriff auf das verteilte Objekt beschreiben.

Mit Hilfe des IDL Compilers können nun bestimmte Interfaces zu einem idl File kompiliert werden. Nun kann der andere Teil der verteilten Anwendung, welcher möglicherweise in einer anderen Sprache geschrieben ist, einfach dieses IDL File zu einem Interface seiner eigenen Sprache (rück)kompilieren.

Dies erfolgt durch ein Mapping welches angibt wie ein Interface in IDL in der eigenen Programmiersprache aussieht.

Dabei gillt:

* Descriptive Sprache
* Ohne Logik
* Nur Syntax

**Interfaces**

Wie auch bei RMI gibt es zwei Möglichkeiten des verteilten Methodenaufrufes. Die statische Möglichkeit so wie die dynamische Möglichkeit.

Bei einem statischen Aufruf werden ausschließlich die Interfaces die durch den IDL Compiler erstellt werden verwendet.

Für dynamischen Aufrufe gibt zusätzliche spezielle **Dynamic Invocation Interfaces.**

Dabei muss mein Methodenaufruf folgendes übergeben werden:

* Objekt Reference
* Operation welche ausgeführt werden soll
* Parameter für die Operation

Usage:

* getInterface(); //objekt.getInterface um InterfaceDef. zu erhalten
* describe\_interface(); //um erhaltene InterfaceDef zu beschreiben
* create\_request(); //object.create\_request(objref,operation,param)
* invoke(),send()

## CORBA Setup

Für unsere Übungen wollen wir zwei ORB’s auf einer virtuellen DEBIAN Maschine installieren.

Dafür verwenden wir für den Java Teil der Übung(Client) **JACORB** und für den C++ Teil der Übung(Server) **OmniOrb.**

### OmniOrb Installation/Setup

**1)**

Um OmniOrb herunterzuladen führen wir im gewünschten Verzeichnis folgendes aus:

*apt-get install* [*https://sourceforge.net/projects/omniorb/files/latest/download?source=files*](https://sourceforge.net/projects/omniorb/files/latest/download?source=files)

**2)**

Nun gehen wir in das heruntergeladene Verzeichnis und erstellen mit   
*mkdir build*einen Ordner in dem wir OmniOrb builden wollen.  
**3)**

Jedoch wird das nicht funktionieren bevor wir nicht noch einige Komponenten die dazu notwendig sind installiert haben.

Also laden wir zunächst wieder mit *apt-get install folgende Pakete runter:*

* *gcc (+suggested packages)*
* *g++ (+suggested packages)*
* *libpython-2.7-dev*

**4)**

Nun können wir das OmniOrb builden. Dazu führen wir im Ordner build folgende Befehle aus:

*../configure, make, make install*

***5)****Nun muss noch ein* ***Naming Service*** *erstellt werden:*

*mkdir /var/omninames welcher später beim Testen wie folgt gestartet wird*

*omniNames -start -always*

### JacORB Installation/Setup

Bei jacORB haben wir uns dazu entschieden nur die Binary’s runterzuladen. Daher fällt das Setup deutlich leichter aus.

Im Verzeichniss in welchen JacORB installiert werden soll führen wir

*apt-get install* [*http://www.jacorb.org/releases/3.7/jacorb-3.7-binary.zip*](http://www.jacorb.org/releases/3.7/jacorb-3.7-binary.zip)

*aus.*

### Ordner /opt

Für einen leichteren Zugriff auf die gebuildeten ORB’s wollen wir diese nochmal extra in einem eigenen Verzeichnis speichern. Dies vereinfacht außerdem den Zugriff der Anwendungen auf die ORB’s.

Also kopieren wir mit dem Befehl cp den Inhalt des des JaCORB Verzeichnisses sowie den Inhalt des /build Verzeichnisses in den Ordner opt.

*cp /SYT/omniorb/build omniorb.versionnmr*

*cp /SYT/jacorb jacorb.versionnmr*

*Desweitern erstellen wir in /opt nochmal zwei Ordner die auf unsere beiden ORB’s verlinken, nur diesmal ohne Versionsnummer.*

*ln -s omniorb.version omniorb*

*ln -s jacorb.version omniorb*



### Zusätzliche Pakete

Für die spätere Ausführung der Programme und die Arbeit mit Git laden wir noch folgende Pakete runter.

*apt-get install git ant openjdk-8-jdk*

## Callback Anwendung

Nachdem beide ORB’s erfolgreich installiert wurden, und dies mit dem von Herr Professor Borko zur Verfügung gestellten Hello World Programm [2] getestet wurde, soll nun eine eigene Anwendung verteilt werden.

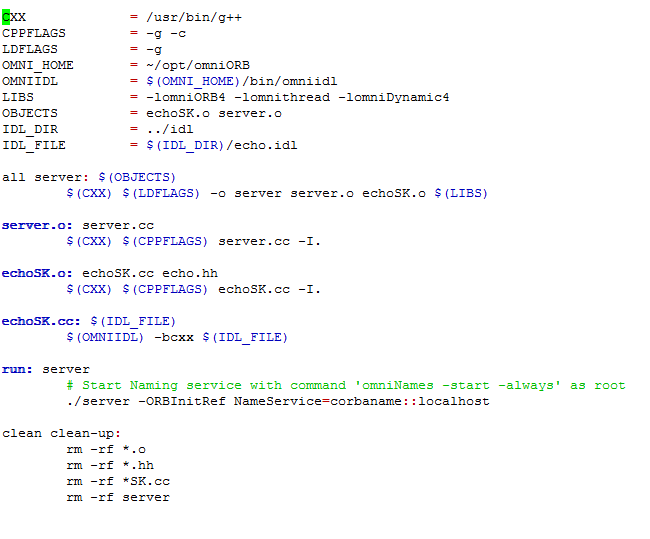
Dazu nehmen wir die callback-server Anwendung aus den Examples von OmniOrb und schreiben unsere eigene Client-java Anwendung dazu.

### Server Seite

Auf der Server Seite unseres verteilten Systems brauchen wir die die c++ Anwendung und ein Makefile in welchem der Pfad zum IDL Compiler und dem ORB angegeben werden.

Da der C++ Code der Anwendung ja vorhanden ist, muss hier nur noch das MakeFile erstellt werden. Als Basis dafür nehmen wir das MakeFile welches wir beim HelloWorld[2] Programm benutz haben.

**Zusatz:** Sollte man das IDL file anders benennen als in den examples muss man den Import am Anfang des C++ Files natürlich ändern.

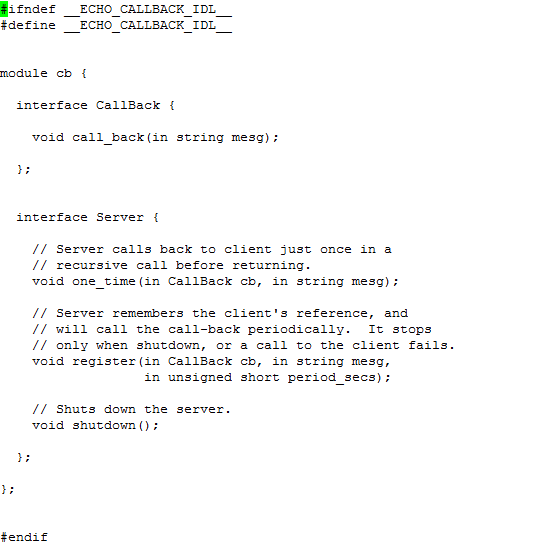


1 Makefile ServerSeite

### IDL File

Natürlich muss auch hier ein IDL-Interface vorhanden sein damit unsere beiden Anwendungen miteinander kommunizieren können.

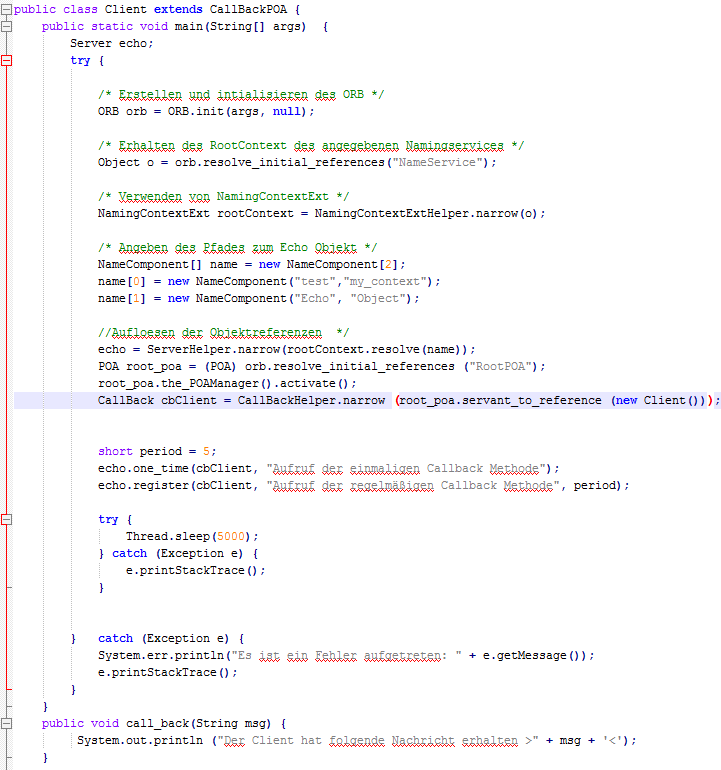
Hier das benötigte IDL File <echo.idl>:



### Client Seite

Nun müssen wir eine passende Java Anwendung und das dazugehörige build.xml erstellen die die gewünschten Methoden des Server Objektes aufruft.

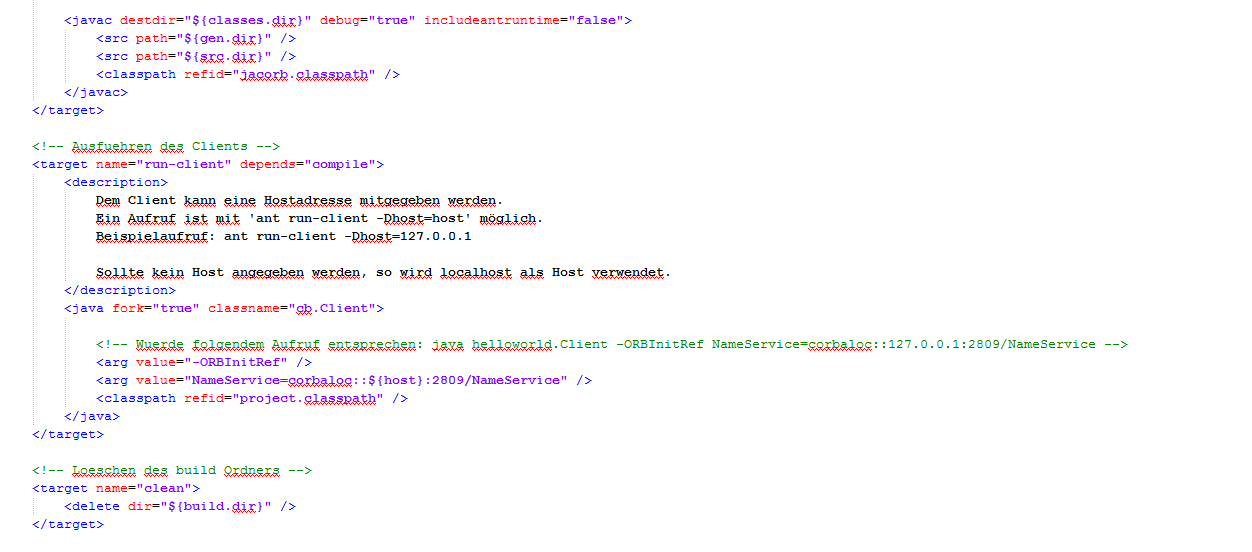
**JAVA Anwendung:**



**build.xml**

Hier nun das build File. Das in den Code-examplen mitgeschickte build.xml wurde wieder als Grundlage verwendet[2].





## Testen

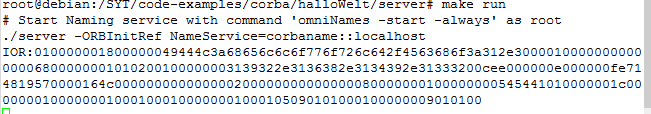
Für das testen einer Anwendung müssen folgende Schritte getätigt werden.

* Terminal 1:Starten des in Kapitel 2.2 beschriebenen NameServices
  + omniNames –start –always
* Terminal 2: Starten der Server anwendung
  + Make run im Server Verzeichniss
* Terminal 3 : Starten der Client Anwendung
  + ant run-client

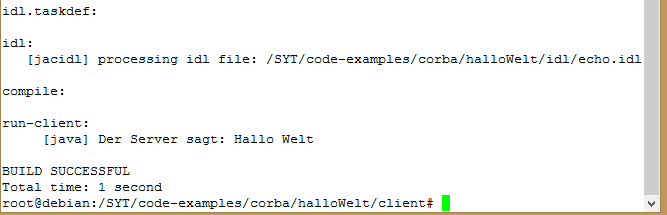
### HelloWorld testen

Zuerst Testen wir das HelloWorld Programm aus den Code Examplen

**Server starten:**

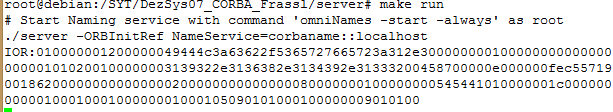


**Client ausführen**

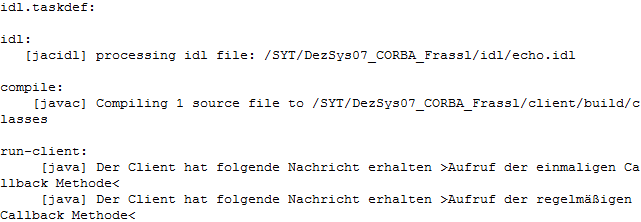


### Callback testen

**Server starten**



**Client starten**



# Quellen / Literatur

"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>

[1]FOLIENSATZ **Distributet-Objects Corba** aus dem Unterricht

[1]Code Examples: <https://github.com/mborko/code-examples.git>

# Zeitaufwand/ Probleme

Die Konfiguration der ORB’s und des Naming Services empfand ich zuerst als sehr unübersichtlich und schwierig. Trotzdem habe ich es dann aufgrund der ausführlichen Erklärung im Unterricht und auch durch Hilfestellung von einem Mitschüler geschafft, die Konfiguration innerhalb der Laborunterrichtszeit zu meistern.

Das Einbauen der beiden Aufaben (HelloWorld,Callback) wurde aufgrund der code-examples und der Beispielprogramme der beiden ORB’s deutlich verständlicher. So konnten bereits vorhandene Files für das eigene Programm angepasst werden. Trotzdem sind mir häufig Fehler in den source- Files und den Makefiles unterlaufen, die es zu beheben galt.

|  |  |  |
| --- | --- | --- |
| Tätigkeit | Zeitaufwand | Datum/Ort |
| ORB’s Konfigurieren  NamingService | 180 | Labor  16.04.2016 |
| HelloWorld Implementieren | 60 | Labor  16.04.2016 |
| Callback Serverseite und IDL  (omni examples) | 60 | Zuhause  19.04.2016 |
| Callback  ClientSeite | 150 | Zuhause  21.04.2016 |
| Protokoll | 300 | Zuhause  22.04.2016 |
| Gesamt | 12.5 Stunden |  |

# GITHUB

<https://github.com/gfrassl-tgm/DezSys07_CORBA.git>