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

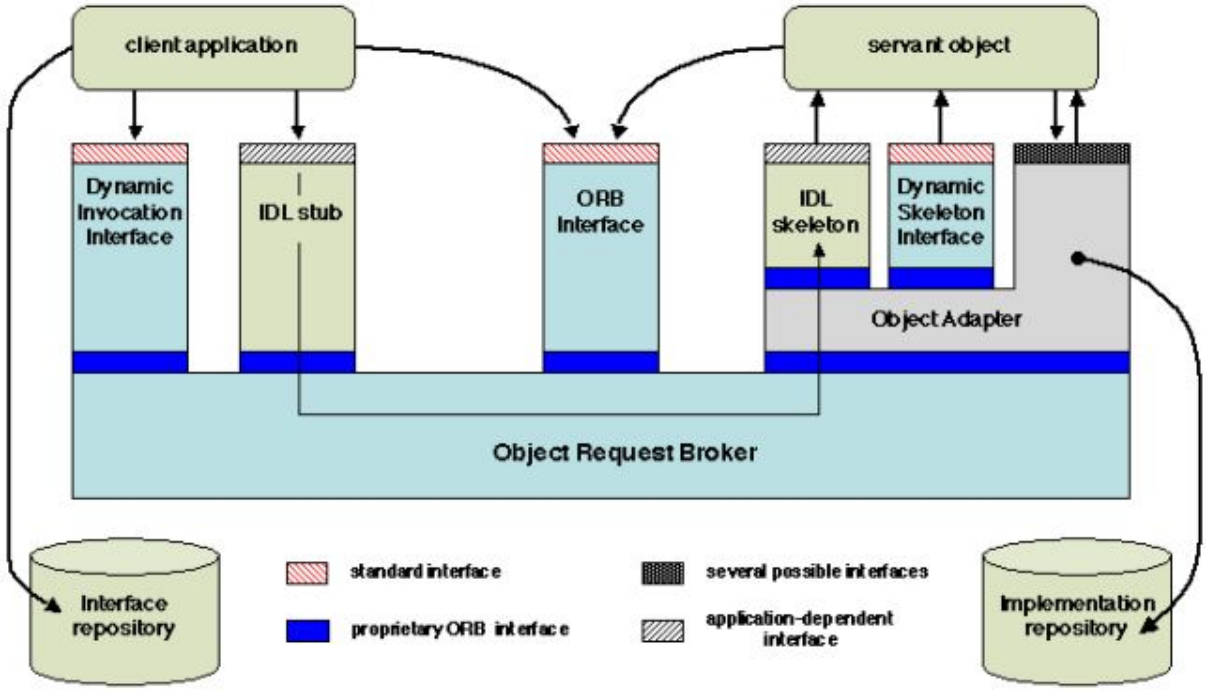
Begonnen am 19.Mai 2016

Beendet am 19.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.



Quelle: S.Krakowiak, „Middleware Architecture with Patterns and Frameworks“
http://sardes.inrialpes.fr/~krakowia/MW-Book/Chapters/DistObj/distobj-body.html

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

“screen”; http://www.debianroot.de/server/linux-screen-debian-1065.html

“IDL”; http://www.cs.rpi.edu/~musser/dsc/idl/idl-overview_5.html

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 Vorlag, beziehungsweise Orientierungshile, verwende ich die jeweils zur Verfügung gestellten Demo-Programme. Zusätzlich habe ich “screen”, via apt-get install screen, installiert um die Notwendigkeit mehrer Terminals zu umgehen.

Dieser Teil unterscheidet sich maßgeblich von dem ursprünglichen Protokoll. Auf Anraten meiner Kollegen habe ich das alte Programm vollständig verworfen und ein neues Calculator Programm geschrieben. Das alte Dokument findet sich bei dem alten Code in meinem GitHub Repository.

3.3.1 IDL File

Das File beinhaltet das module “calc” indem das Interface “Calculator” definiert wird. Dieses beinhaltet die Exception DivideZeroException, sowie die Methoden zur Durchführung der Grundrechenarten mul,sub,add,div. Div wirft (mit “raises” in IDL bezeichnet) diese oben erwähnte DivideZeroException.

3.3.2 Server Seite

Makefile

Das Makefile hätte nicht angepasst werden müssen, allerdings habe ich zur Erhöhunh der Überischtlichkeit jedes “echo” durch “calculator” und jedes “Echo” durch “Calculator” ersetzt, damit die auto-generierten Files nicht mehr echoSK.o & Co, sondern calculatorSK.o heißen

Server.cc

Der Server muss daher anstatt echo.hh nun calculator.hh inkludieren. Außerdem habe ich die Methode echoString entfernt und dafür die Methode add,sub,div und mul erstellet. Alle haben 2 CORBA::Long Parameter und bis auf div, welche einen CORBA::Double als Rückgabewert hat, geben auch alle einen CORBA::Long zurück. Der Namespace “CORBA” muss vor den Datentypen angegeben werden, da CORBA ein Sprachenunabhängiger Standard ist, die Sprachen selbst allerdings unterschiedlich viel Speicherplatz für ihre Datentypen brauchen und es dadurch, gäbe es diese Vereinheitlichung nicht, zu Problemen kommen könnte. Im Gegensatz zu Java kann man

in C++, aufgrund der Überladung von Operaten, diese komplexen Datentypen auch mit normalen Rechenzeichen verwenden, ohne eine valueOf() oder ähnliche Methoden zu benötigen. Die div Methode überprüft zusätzlich ob der Divident oder der Divisor 0 ist und wirft, wenn dem so ist, eine selbst erstellte DivideZeroException die vom Java Client aufgefangen wird.

3.3.3 Client Seite

build.xml

Client.java

Hier werden nun die Methoden des Servers verwendet. Dafür waren einige Imports der neuen Methoden notwendig. Außerdem instanziiert man einen Calculator mittels der CalculatorHelper.narrow(rootContext, resolve(name)) Methode. “name” bezieht sich auf das erstellte NameComponent Objekt das an der Stelle 0 einen Testeintrag (“test”,”my_context”) und an der Stelle 1 “Calculator” und “Object” enthällt. “rootContext” ist das NamingContextExt Objekt das über die NaminContextExtHelper.narrow(o) Methode instaziiert wurde. Darüber hinaus wird ein ORB Objekt über die Methode ORB.init(args,null) instanziiert, das zum Erhalt des RootContext gebraucht wird. Hierfür nutzt man orb.resolve_initial_references(“NamingService”);.

3.3.4 Testen des Programms

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

```
root@debian:/home/zeljko/repos/code-examples/corba/dezsys07/client# ant run-client
Buildfile: /home/zeljko/repos/code-examples/corba/dezsys07/client/build.xml

idl.taskdef:

idl:
   [jacidl] processing idl file: /home/zeljko/repos/code-examples/corba/dezsys07/idl/calculator.idl

compile:
    [javac] Compiling 1 source file to /home/zeljko/repos/code-examples/corba/dezsys07/client/build/classes

run-client:
     [java] 5 * 5 = 25
     [java] 5 - 5 = 0
     [java] 5 + 5 = 10
     [java] Es ist ein Fehler aufgetreten weil die Division mit 0 nicht definiert ist.

BUILD SUCCESSFUL
Total time: 1 second
```

Die Ausgabe wird vom Java Client generiert indem er die Methoden des C++ Servers aufruft und ihr Ergebnis, mit der Aufgabenstellung, ausgibt. Die letzte Zeile zeigt den Output der DivideZeroException. Die Angabe ist momentan hardcoded, ließe sich aber über eine einfache Parameterübergabe bei Aufruf des Clients oder eine Abfrage über die Kommandozeile leicht usergesteuert erweitern.

4 Zeit und Probleme

4.1 Probleme

Dieser zweite Anlauf war auf Anhieb um einiges problemärmer, da ich mich bereits an das Programmieren in der Konsole gewöhnt habe und die bereits angeeigneten Grundlagen gut umsetzen konnte. Ein Fehler ist mir allerdings doch untergekommen.

Java- Exception “ClassDefNotFound : ReadInputstream” in JacORB-Library

Nach reichlich Recherche hat sich heraus gestellt, dass dieser Fehler auftritt wenn von der C++ Implementierung eine Exception ausgelöst und von der Java Implementierung aufgefangen wird, weil in der ant Datei die Librarys zwar beim Kompilieren geladen werden, beim ausführen allerdings nicht. Beheben lässt sich dies indem man im build.xml die Zeile :

<classpath refid=”jacorb.classpath” />

von <!-- Kompilieren des Quellcodes → kopiert und sie bei <!-- Ausfuehren des Clients --> nochmals anfügt.

Zeitangaben	Schätzung	Aufwand
Stunden	5	3