Gruppe Pawlowsky, Sochovsky

Implementierung einer CORBA Applikation

Caesar-Verschlüsselung

VSDB 4BHITS 02.03.2012

Pawlowsky, Sochovsky

Inhalt

Aufgabenstellung 2

Punkte 2

Resources 2

Zusätzliches 3

CORBA Installation 4

OB Orbacus für C++ 4

JOB Orbacus für Java 4

How to compile your IDL 5

How to import it into an IDE 5

Visual Studio C++ 5

Eclipse JAVA 5

Start der Applikation 6

Start des Nameservice 6

Start des C++ Servers 6

Start des Java-Clients 6

Erklärung der Ausführung 6

Designüberlegung 7

Arbeitsteilung 9

Arbeitsdurchführung 10

Tests 12

Lokale Tests: 12

Netzwerk Tests: 13

Quellen 14

Aufgabenstellung

Implementieren Sie den Cäsar-Verschlüsselungsalgorithmus [1] als eine CORBA-Applikation. Halten Sie sich an das Tutorial von Carlos Jiménez de Parga [2]. Beachten Sie dabei, dass Sie entweder den Client oder den Server in Java implementieren müssen. Sie können dabei entweder die im Beispiel verwendete ORBacus [3] CORBA-Implementierung verwenden, oder auf omniORB [4] und JacORB [6] ausweichen. Dabei ist omniORB als Paket im Standard Repository von Ubunut [5]/Debian enthalten und sehr einfach zu installieren.

Bedenken Sie bitte, dass der beliebte "Versuch und Irrtum-Ansatz" hier zu sehr hohem Zeiteinsatz führen würde. Daher bitten wir um eine grundlegende Recherche, ***bevor*** mit der Implementierung begonnen wird [7,8,9,10].

Halten Sie sich an die Metaregeln, und dokumentieren Sie in Ihrem Protokoll bitte die grundlegenden Technologieansätze als auch die wichtigen Codestellen sowie alle notwendigen Schritte Ihr Programm zu laufen zu bringen. Sie werden diese Informationen auf jeden Fall noch einmal benötigen! Vergessen Sie ebenso nicht auf die Inline-Codekommentare.

Diese Aufgabenstellung ist als 2er-Team zu lösen.

Punkte

Dokumentation der CORBA Komponenten (IDL/-Compiler, Nameservice, POA/-Manager, Verwendung vom ObjectRequestBrokers) [4Pkt]
Implementierung des C++ Tutorials [4Pkt]
Implementierung mit Java und C++ [4Pkt]
Ausführung und Tests in der heterogenen Umgebung (Java/C++) [4Pkt]

Resources

[1] Caesar-Verschlüsselung; Wikipedia; Letzte Änderung am 21. Februar 2012; Online:http://de.wikipedia.org/wiki/Caesarverschl%C3%BCsselung; abgerufen 1.3.2012
[2] A Simple C++ Client/Server in CORBA; Carlos Jiménez de Parga; Codeproject; September 2009; Online:http://www.codeproject.com/Articles/24863/A-Simple-C-Client-Server-in-CORBA; abgerufen 1.3.2012
[3] ORBacus; Progress Software; Online:http://www.progress.com/en/orbacus/index.html; abgerufen 1.3.2012
[4] omniORB; Online:http://omniorb.sourceforge.net/index.html; abgerufen 1.3.2012
[5] Search result vor omniorb; ubuntu.com; Online:http://packages.ubuntu.com/search?keywords=omniorb&searchon=names&suite=oneiric&section=all; abgerufen 2.3.2012
[6] JacORB; Online: http://www.jacorb.org; abgerufen 1.3.2012
[7] Documents Associated with CORBA, 3.2; OMG; November 2011; Online:http://www.omg.org/spec/CORBA/3.2/; abgerufen 1.3.2012
[8] Orbacus Product Documentation; D.Pierce; Progress Software; Dezember 2011; Online:http://communities.progress.com/pcom/docs/DOC-105219; abgerufen 1.3.2012
[9] JacORB Programming Guide 2.3; JacORB Team; Februar 2007; Online:http://www.jacorb.org/releases/2.3.0/ProgrammingGuide.zip; abgerufen 1.3.2012
[10] Online:http://omniorb.sourceforge.net/docs.html; abgerufen 1.3.2012

Zusätzliches

Es wurde mit dem für die Aufgabe zuständigen Lehrer bei der Erklärung der Aufgabe explizit besprochen das bei dieser Implementierung einer CORBA Applikation nicht darauf zu achten ist ob das Programm eine Excepition wirft oder andere Probleme aufwirft. Es soll einfach nur unter den Umständen für die es programmiert wurde funktionieren. Außerdem durften wir bei der Art des Corbaproduktes selbstentscheiden welches wir nehmen. Wir haben uns für Obacus entschieden. Zusätzlich durfte man es sich aussuchen in welcher Sprache der Server oder der Client geschrieben wird.

CORBA Installation

OB Orbacus für C++

1. „perl“ und „nmake“ installieren
    a. „nmake“ erhält man über die Installation von Visual Studio oder WindowsSDK
    b. „perl“ dieses Programm erhält man von der Seite perl.org
        i. Wir benutzen ActiveState Perl v5.14.2
2. Nun muss man die „perl“ PATH variable setzen
    a. Dies muss man nicht immer tun, es kann passieren das es bei der Installation nicht passiert
3. „runconfig .bat“ ausführen vom OB
    a. Diese bat Datei befindet sich in dem entpackten Ordner von OB
    b. Die bat Datei schlägt bereits mögliche Konfigurationen vor (2 Abweichungen)
        i. shared Libraries auf “no” setzen
        ii. einen eigenen Pfad wählen um Ob in den richtigen Ordner zu kompilieren

WICHTIG ALLE FOLGENDEN EINGABEN MÜSSEN IN DER SELBEN KONSOLE DURCHGEFÜHRT WERDEN!

4. vcvars32.bat ausführen (Ordner: Visual Studio/VC/bin/)
5. mittels „cd“ in den Ordner von „nmake“ gehen
    a. dann „nmake /f (ganzer Pfad zu)makefile.mak“
        i. diese Zeile kompiliert das Programm danach muss man noch
    b. Danach „nmake /f makefile.mak install oder install_min“
        i. Nun wird Orbacus auch installiert
        ii. „install_min“ installiert die Beispielprogramme nicht
6. danach noch “nmake /f makefile.mak clean“ das löscht die temporären Dateien
7. man muss in die Umgebungsvariable PATH den installierten Orbacus Ordner (OOC)/bin hinzufügen
8. Im heruntergeladenen File(Caesar Server + Client in C++) gibt es eine NameServ.exe diese muss man mit der im installierten normalen OB/bin Ordner befindlichen gleichnamigen Datei ersetzen

JOB Orbacus für Java

1. „ANT“ runterladen (es gibt mehrere ANT Versionen wir benutzten die von Apache)
2. wenn man „ANT“ hat ist darin ein File „ant“ ohne Dateiendung!!!!!
3. und dann folgenden Befehl (man muss aber im JOB Pfad sein mittels „cd“) ausführen
    a. ant –Dinstall.dir=“/gewünschtes Verzeichnis von JOB“
4. den Pfad vom gewünschten Verzeichnis/bin muss man dann in die benutzerweite PATH Variable setzen
5. Dann falls ein Fehler mit der tools.jar(im Ordner vom JDK) kommt muss man das JAVA home directory als JAVA_HOME (in der Konsole schreiben: set JAVA_HOME="Pfad zum JDK Ordner")
6. oder man muss sowohl in die CLASSPATH Variable als auch in die PATH Variable das JDK Directory hinzufügen

7. danach könnte der Fehler „* cannot include itself“ auftreten
    a. wenn der kommt muss eine Zeilennummer dabeistehen
    b. mit der Zeilennummer muss man im „BUILD.xml“ dieser Zeile die „boolean“ Variable auf „false“ setzen → Dieser Fehler ist bei uns nicht aufgekommen -

How to compile your IDL

1. Man muss sein IDL File schreiben (oder kopieren (Angabe))
2. Nun sucht man im JOB / OB Ordner nach der Datei IDL.exe
    a. Je nachdem welche Programmiersprache man möchte
3. Nun führe in der Konsole “IDL.exe pathtofile.idl”
    a. Oder für JAVA muss man die „jidl.exe“ benutzen

How to import it into an IDE

Visual Studio C++

1. Mit dem erstellten IDL interface kann man nun ein neues Projekt starten
2. Project + Rechts-Klick + properties
    ➔ Beim “linker” "zusätzliche Bibliotheksverweise" füge „OBdirectory/lib“ hinzu
    ➔ Im OBdirectory/include sind 2 Ordner einer ist OB und der andere ist JTC
        i. Beide muss man in den Compilerordner von Visual Studio kopieren
            1. visualStudioPath/VC/include /
3. Bei uns gab es dann einen spezifischen Fehler der bei anderen Gruppen nicht aufgetreten ist
    ➔ In den Orbacus Files steht eine long Variable die leider falsch befüllt wird (in jeder Datei)
    ➔ Man muss manuell in jeder Datei den long Wert auf 4030400L ändern
4. Wenn man die vom idl Compiler generierten Files ins Projekt lädt dann kann es passieren, dass die IDE in jeder Zeile der Files einen Fehler anzeigt wenn das passiert erstellt man ein neues File mit dem gleichen Namen und kopiert den Inhalt neu hinein
    ➔ Dies kann auch unter Eclipse passieren

Eclipse JAVA

1. Man erstellt auch hier ein neues Projekt mit den erstellten IDL-Stubs
    a. keine weiteren Abweichungen mit Problemen bekannt

Start der Applikation

Start des Nameservice

Nameserv.exe -OAhost localhost -OAport 8140

Start des C++ Servers

Server.exe -ORBInitRef NameService=corbaloc:iiop:localhost:8140/NameService

Start des Java-Clients

Java –jar client.jar -ORBInitRef NameService=corbaloc:iiop:localhost:8140/NameService

Erklärung der Ausführung

- -ORBInitRef NameService bedeutet, dass die nachfolgenden Anweisungen für den ORB weitergegeben wird und das der das verwendet
- NameService = gibt dann sozusagen den Pfad zum Nameservice an
- iiop ist dann das Protokoll das verwendet wird um zu kommunizieren
- /NameService ist der Name des Services das angesprochen wird

Beim Start des Nameservice musste man den Port angeben dieser darf frei variiert werden, jedoch müssen Server und Client ihn kenn und dann zum Hostnamen oder zur IP-Adresse schreiben
Mit dem Abschnitt -OAhost localhost hängt man diesen Nameservice an seine eigene IP

In Client und Server gibt es den Abschnitt:
iiop:localhost:8140
in diesen Teil darf man einen anderen Hostnamen oder eine andere IP-Adresse einsetzen, auch der Port kann hier verändert werden, jedoch muss man immer den Nameservice ansprechen.

Es ist notwendig, dass man auf jeden Fall den Nameservice als Erstes startet.

Um diese 3 Teilprogramme lauffähig auszuführen, darf man als Server sich auf keinen Fall mit den NameService als localhost verbinden sondern gibt auch wenn man auf der gleichen Maschine ist seine IP an.

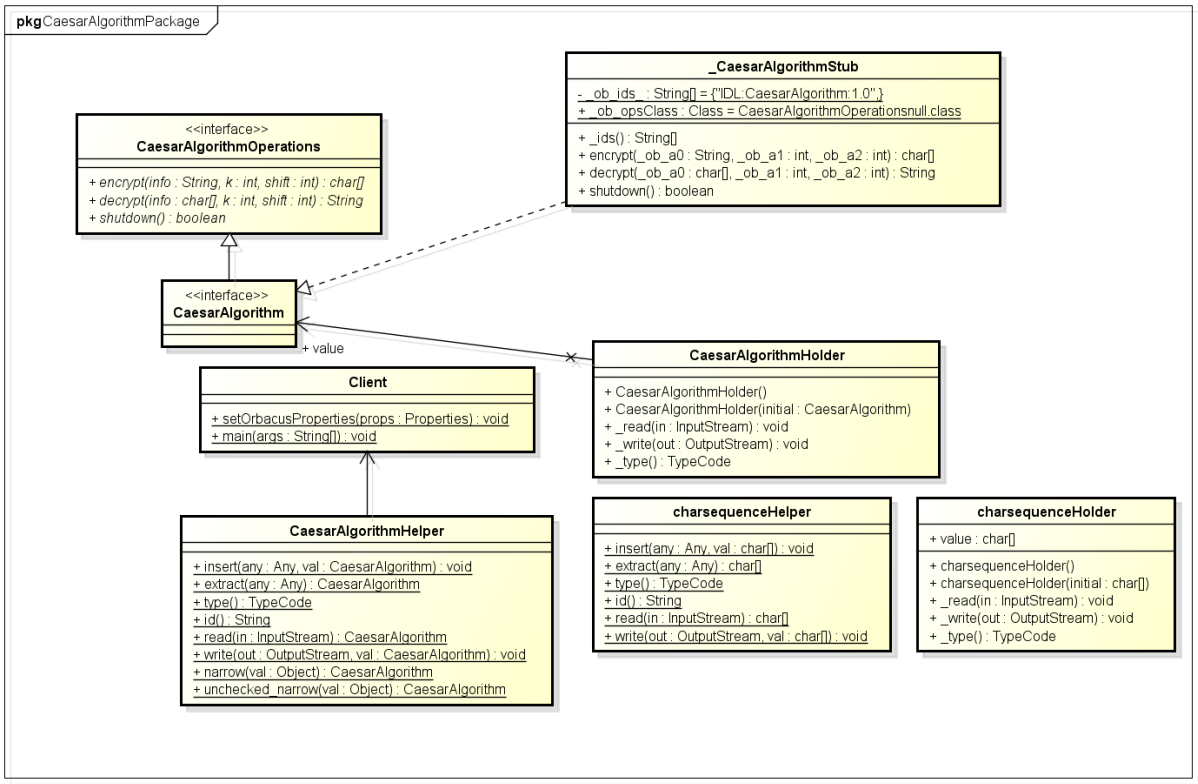
Designüberlegung

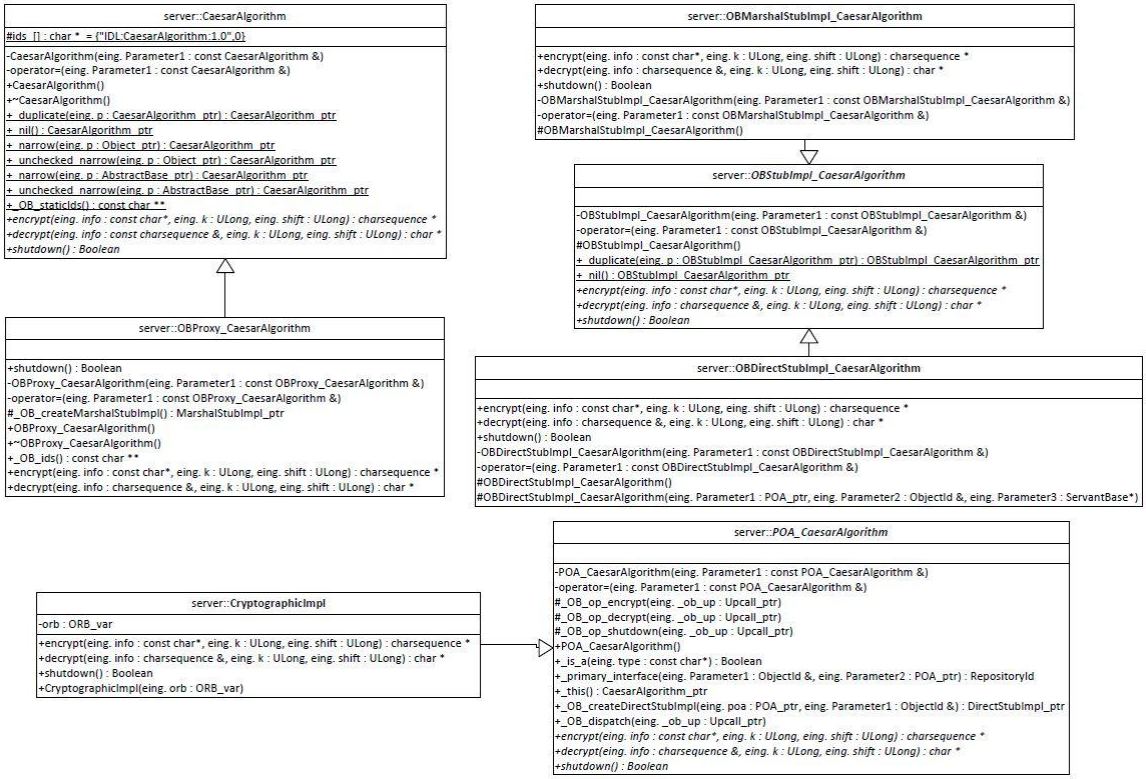
Bei der grundsätzlichen Designüberlegung haben wir uns möglichst stark an die Vorgabe gehalten und diese auch eingehalten.
Es gibt die vorgegebenen C++ Dateien die wir nicht weiter bearbeitet haben, wir haben nur das dazugehörige IDL File generieren lassen.
Auch das IDL File haben wir aus der Angabe genommen.

Bei der Implementation mit JAVA haben wir genau eine Klasse erstellt die den Client repräsentiert. Außerdem musste man auch ein CORBAInterface das wir mit der Vorgabe.idl generiert haben in dem Client einbinden.

Im Prinzip haben wir den C++ Client 1-zu-1 in JAVA „übersetzt“.

- Bedeutet selbe ausgaben
- Unterschiedliche Aufrufe von CORBA
- Gleiche Übergabeparameter




server::CaesarAlgorithm
#ids_[] : char * = {"IDL:CaesarAlgorithm:1.0",0}
-CaesarAlgorithm(eing. Parameter1 : const CaesarAlgorithm &)
-operator=(eing. Parameter1 : const CaesarAlgorithm &)
+CaesarAlgorithm()
+~CaesarAlgorithm()
+_duplicate(eing. p : CaesarAlgorithm_ptr) : CaesarAlgorithm_ptr
+_nil() : CaesarAlgorithm_ptr
+_narrow(eing. p : Object_ptr) : CaesarAlgorithm_ptr
+_unchecked_narrow(eing. p : Object_ptr) : CaesarAlgorithm_ptr
+_narrow(eing. p : AbstractBase_ptr) : CaesarAlgorithm_ptr
+_unchecked_narrow(eing. p : AbstractBase_ptr) : CaesarAlgorithm_ptr
+_OB_staticIds() : const char **
+encrypt(eing. info : const char*, eing. k : ULong, eing. shift : ULong) : charsequence *
+decrypt(eing. info : const charsequence &, eing. k : ULong, eing. shift : ULong) : char *
+shutdown() : Boolean
server::OBProxy_CaesarAlgorithm
+shutdown() : Boolean
-OBProxy_CaesarAlgorithm(eing. Parameter1 : const OBProxy_CaesarAlgorithm &)
-operator=(eing. Parameter1 : const OBProxy_CaesarAlgorithm &)
#_OB_createMarshalStubImpl() : MarshalStubImpl_ptr
+OBProxy_CaesarAlgorithm()
+~OBProxy_CaesarAlgorithm()
+_OB_ids() : const char **
+encrypt(eing. info : const char*, eing. k : ULong, eing. shift : ULong) : charsequence *
+decrypt(eing. info : charsequence &, eing. k : ULong, eing. shift : ULong) : char *
server::OBMarshalStubImpl_CaesarAlgorithm
+encrypt(eing. info : const char*, eing. k : ULong, eing. shift : ULong) : charsequence *
+decrypt(eing. info : charsequence &, eing. k : ULong, eing. shift : ULong) : char *
+shutdown() : Boolean
-OBMarshalStubImpl_CaesarAlgorithm(eing. Parameter1 : const OBMarshalStubImpl_CaesarAlgorithm &)
-operator=(eing. Parameter1 : const OBMarshalStubImpl_CaesarAlgorithm &)
#OBMarshalStubImpl_CaesarAlgorithm()
server::OBStubImpl_CaesarAlgorithm
-OBStubImpl_CaesarAlgorithm(eing. Parameter1 : const OBStubImpl_CaesarAlgorithm &)
-operator=(eing. Parameter1 : const OBStubImpl_CaesarAlgorithm &)
#OBStubImpl_CaesarAlgorithm()
+_duplicate(eing. p : OBStubImpl_CaesarAlgorithm_ptr) : OBStubImpl_CaesarAlgorithm_ptr
+_nil() : OBStubImpl_CaesarAlgorithm_ptr
+encrypt(eing. info : const char*, eing. k : ULong, eing. shift : ULong) : charsequence *
+decrypt(eing. info : charsequence &, eing. k : ULong, eing. shift : ULong) : char *
+shutdown() : Boolean
server::OBDirectStubImpl_CaesarAlgorithm
+encrypt(eing. info : const char*, eing. k : ULong, eing. shift : ULong) : charsequence *
+decrypt(eing. info : charsequence &, eing. k : ULong, eing. shift : ULong) : char *
+shutdown() : Boolean
-OBDirectStubImpl_CaesarAlgorithm(eing. Parameter1 : const OBDirectStubImpl_CaesarAlgorithm &)
-operator=(eing. Parameter1 : const OBDirectStubImpl_CaesarAlgorithm &)
#OBDirectStubImpl_CaesarAlgorithm()
#OBDirectStubImpl_CaesarAlgorithm(eing. Parameter1 : POA_ptr, eing. Parameter2 : ObjectId &, eing. Parameter3 : ServantBase*)
server::CryptographicImpl
-orb : ORB_var
+encrypt(eing. info : const char*, eing. k : ULong, eing. shift : ULong) : charsequence *
+decrypt(eing. info : charsequence &, eing. k : ULong, eing. shift : ULong) : char *
+shutdown() : Boolean
+CryptographicImpl(eing. orb : ORB_var)
server::POA_CaesarAlgorithm
-POA_CaesarAlgorithm(eing. Parameter1 : const POA_CaesarAlgorithm &)
-operator=(eing. Parameter1 : const POA_CaesarAlgorithm &)
#_OB_op_encrypt(eing. _ob_up : Upcall_ptr)
#_OB_op_decrypt(eing. _ob_up : Upcall_ptr)
#_OB_op_shutdown(eing. _ob_up : Upcall_ptr)
+POA_CaesarAlgorithm()
+_is_a(eing. type : const char*) : Boolean
+_primary_interface(eing. Parameter1 : ObjectId &, eing. Parameter2 : POA_ptr) : RepositoryId
+_this() : CaesarAlgorithm_ptr
+_OB_createDirectStubImpl(eing. poa : POA_ptr, eing. Parameter1 : ObjectId &) : DirectStubImpl_ptr
+_OB_dispatch(eing. _ob_up : Upcall_ptr)
+encrypt(eing. info : const char*, eing. k : ULong, eing. shift : ULong) : charsequence *
+decrypt(eing. info : charsequence &, eing. k : ULong, eing. shift : ULong) : char *
+shutdown() : Boolean

Aufwandsabschätzung

Protokoll:2h
Dokumentation der CORBA Komponenten: 4h
Implementierung des C++ Tutorials: 2h (Server)
Implementierung mit Java: 1h (Client)
Tests: 2h

Gesamt: 9h

Arbeitsteilung

Pawlowsky:

- Protokoll kontrollieren
- Dokumentation der CORBA Komponenten
- Implementierung des C++ Tutorials
- Implementierung mit Java
- Tests

Sochovsky:

- Protokoll schreiben
- Dokumentation der CORBA Komponenten
- Implementierung des C++ Tutorials
- Tests

Arbeitsdurchführung

Sochovsky:

Freitag 02.03.2012: Aufwandsabschätzung 0,5h
Samstag 10.03.2012: Dokumentation der CORBA Komponenten 2h
Samstag 10.03.2012: Implementierung des C++ Tutorials 2h
Montag 12.03.2012: Implementierung des C++ Tutorials 2h
Freitag 16.03.2012: Implementierung des C++ Tutorials 2h
Donnerstag 15.03.2012: Dokumentation der CORBA Komponenten 2h
Donnerstag 22.03.2012: Test und Protokoll anfertigen 2h
Gesamt: 12,5h

Pawlowsky:

Samstag 10.03.2012: Dokumentation der CORBA Komponenten 2h
Samstag 10.03.2012: Implementierung des C++ Tutorials 2h
Sonntag 11.03.2012: Implementierung des C++ Tutorials 3h
Montag 12.03.2012: Implementierung des C++ Tutorials 2h
Mittwoch 14.03.2012: Implementierung des C++ Tutorials 1h
Freitag 16.03.2012: Implementierung des C++ Tutorials 2h
Samstag 18.03.2012: Implementierung des C++ Tutorials 2h
Sonntag 18.03.2012: Implementierung des C++ Tutorials 2h
Sonntag 18.03.2012: Implementierung mit Java 1h
Donnerstag 22.03.2012: Test und Protokoll kontrollieren/verbessern 2h
Gesamt: 18h

Gesamt: 30,5h

Resultate/Niederlagen

Pawlowsky & Sochovsky:

Probleme bei der Installation von Orbacus C++

Lösungen:

Befinden sich in der Anleitung: „CORBA Installation“

Pawlowsky:

Probleme bei der Installation von Orbacus JAVA

Lösung:

Befinden sich in der Anleitung: „CORBA Installation“

Pawlowsky & Sochovsky:

Programmieren mit IDE´s

Lösung:

Befindet sich auch in der Anleitung zur Installation

Pawlowsky & Sochovsky:

Testen übers Netzwerk

Lösung:

Befindet sich in der Erklärung der Ausführung

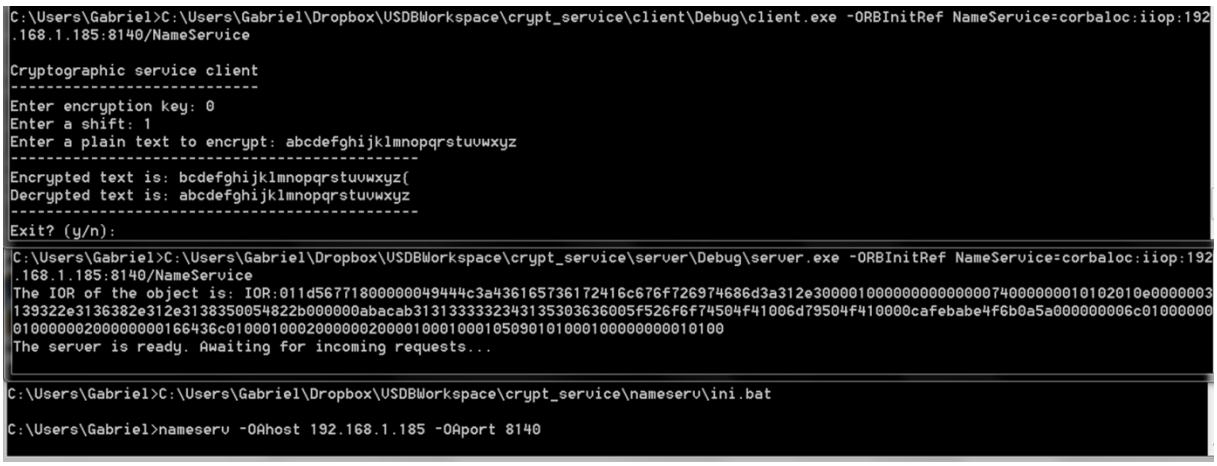
Tests

Testüberlegung:

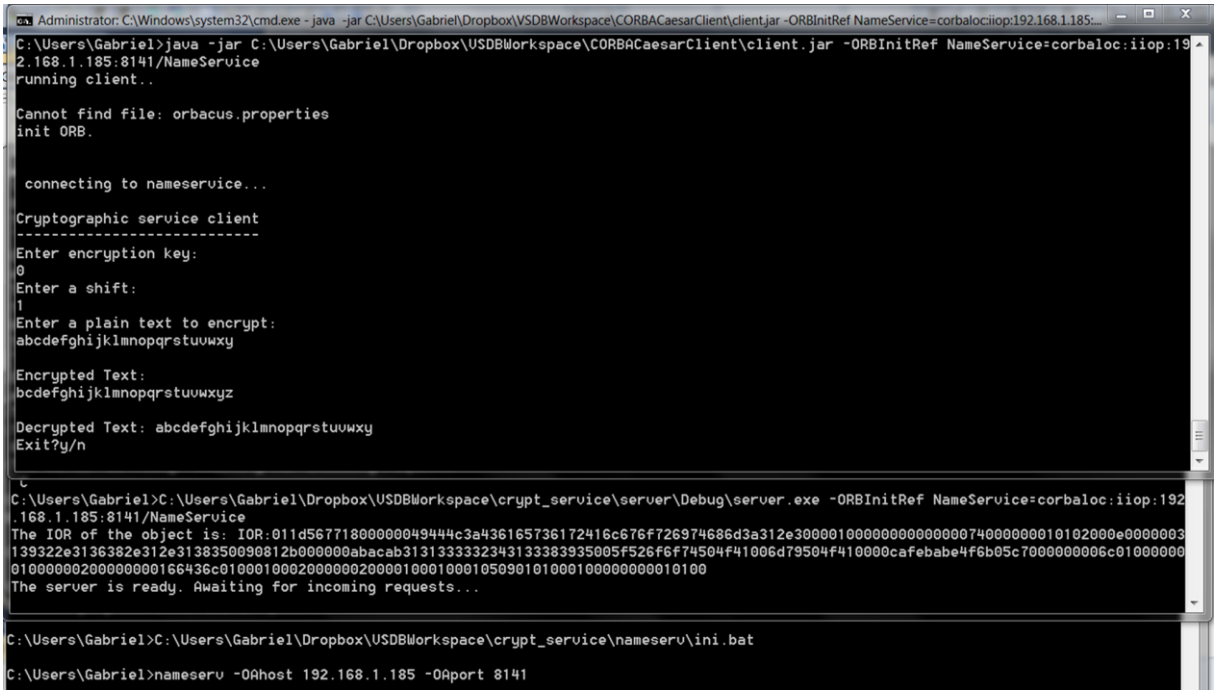
Wir führen mehrere Tests durch, diese teilen sich auf in lokale Tests und Netzwerktests.
Außerdem ist anzumerken das wir auch einen Test mit der C++ Implementation des Clients gemacht haben um noch einmal explizit darauf hinzuweisen das wir Tutorial auch wirklich vollständig durchgeführt.
Worauf zu achten ist wenn man dieses 3er Gespann an Programmen (Client, Server und Nameservice) ausführen möchte ist unter dem Punkt: Erklärung der Ausführung

Lokale Tests:

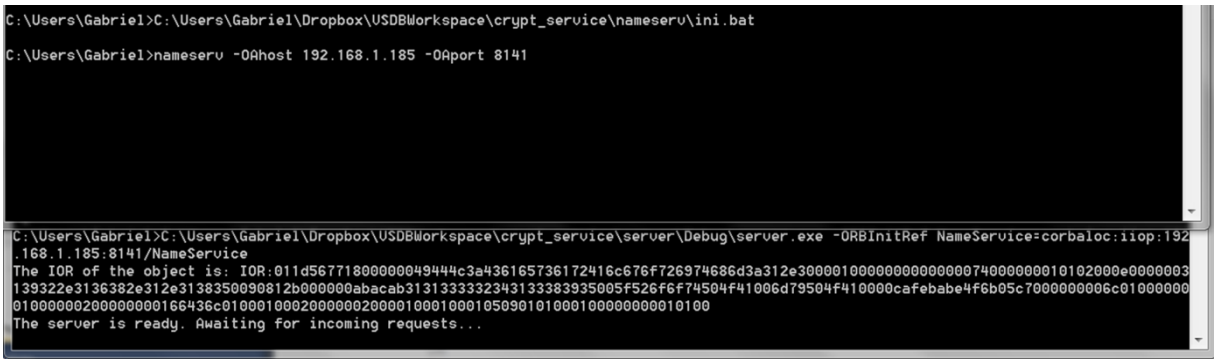
C++

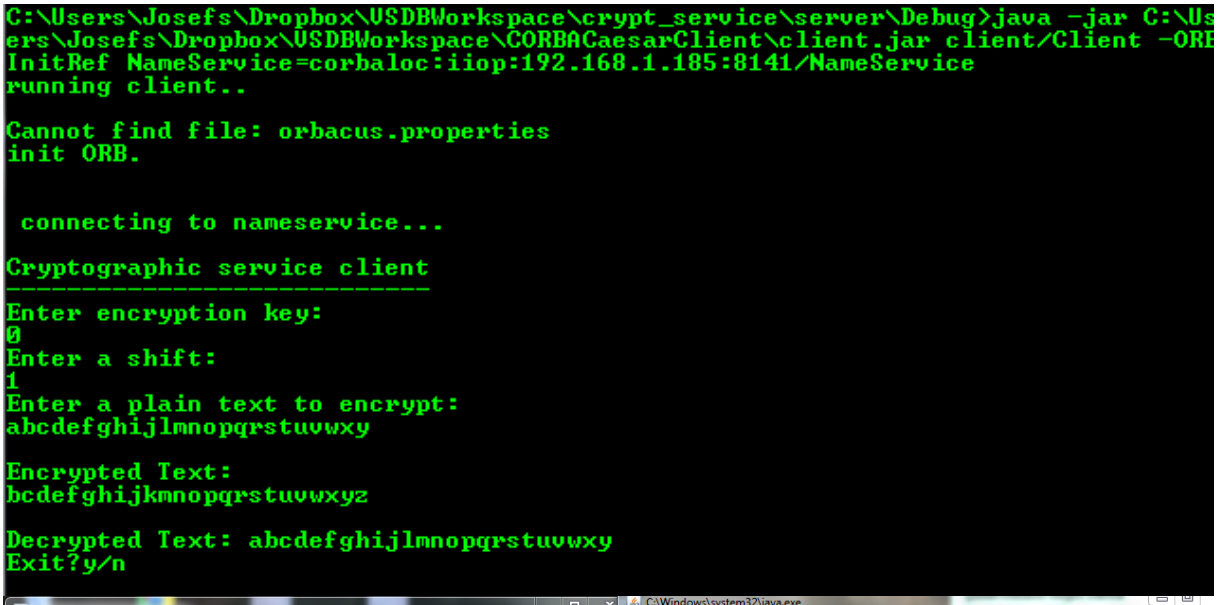


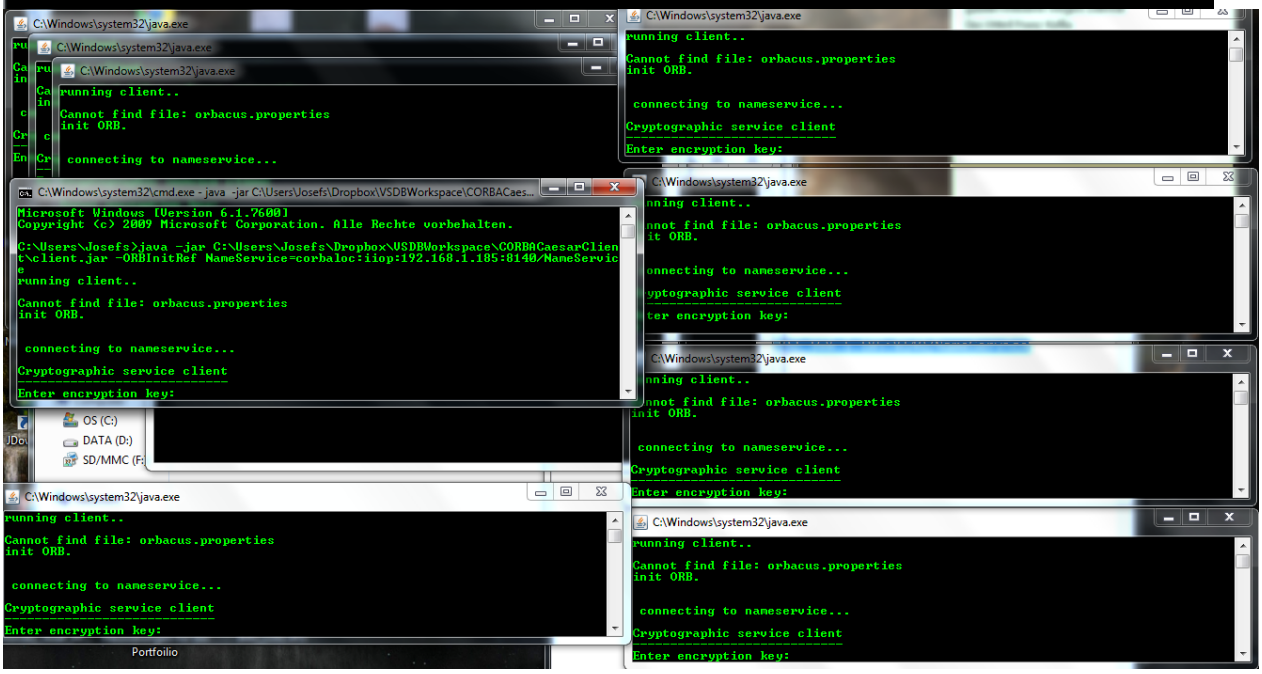
Java



Netzwerk Tests:







Quellen

[1] Documents Associated with CORBA, 3.2; OMG; November 2011; Online:http://www.omg.org/spec/CORBA/3.2/; abgerufen 1.3.2012

[2] ORBacus; Progress Software; Online:http://www.progress.com/en/orbacus/index.html; abgerufen 1.3.2012

[3] A Simple C++ Client/Server in CORBA; Carlos Jiménez de Parga; Codeproject; September 2009; Online:http://www.codeproject.com/Articles/24863/A-Simple-C-Client-Server-in-CORBA; abgerufen 1.3.2012

[4] Practical tutorial for using Corba http://staff.science.uva.nl/~jvgemert/pub/learncorba.pdf zuletzt abgerufen am 22.03.2012