

Netzprogrammierung 7. CORBA II

Prof. Dr.-Ing. Robert Tolksdorf
Freie Universität Berlin
Institut für Informatik
Netzbasierte Informationssysteme
mailto: tolk@inf.fu-berlin.de
http://www.robert-tolksdorf.de



[1] © Robert Tolksdorf, Berlin

Überblick

1. ORB
2. POA
3. NameService
4. Ausführung
5. Persistente Serverobjekte

[2] © Robert Tolksdorf, Berlin

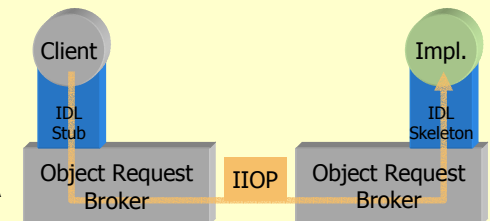
2. Beispiel

[nach <http://developer.java.sun.com/developer/technicalArticles/releases/corba/>]

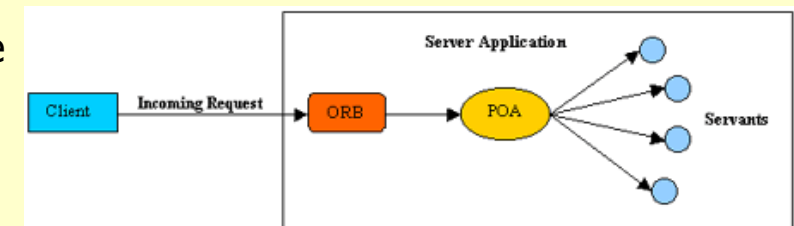
[3] © Robert Tolksdorf, Berlin

Object Adapters

- Konzeptionell:
- Zur Anpassung von Objekten sieht CORBA Objektadapter als Schnittstelle zwischen Programm und ORB vor
- Basic Object Adapter (BOA) bis CORBA 2.2



- Aktuell:
Portable Object Adapter (POA)



[4] © Robert Tolksdorf, Berlin

Anwendung anbinden

- Eine Anwendung muss am Anfang
 - das Laufzeitsystem ORB kontaktieren
 - sich eine Referenz auf den POA geben lassen
 - Teile des POA aktivieren
- Code (praktisch immer gleich):

```
ORB orb = ORB.init(argv, null);
POA rootPOA = POAHelper.narrow(
    orb.resolve_initial_references("RootPOA"));
rootPOA.the_POAManager().activate();
```
- ORB ist aus org.omg.CORBA.ORB
- POA ist aus dem Paket org.omg.PortableServer

[5] © Robert Tolksdorf, Berlin

POA

- POA folgt bestimmten Strategien/Policies zur
 - Verarbeitung von Threads
 - Dauerhaftigkeit von Objekten
 - Objekt IDs
 - Speicherung von Objektverweisen
 - Objektaktivierung
- Eine Anwendung kann sich einen eigenen POA schaffen, der eine neue Policy befolgt

```
POA poa = rootPOA.create_POA("childPOA",
    null, policy);
```
- Details zu Policies siehe Paket org.omg.PortableServer und später bei Persistenz

[6] © Robert Tolksdorf, Berlin

Vektorenaddierer

- IDL einer Beispielanwendung:

```
module ArithApp {
    interface Add {
        const unsigned short SIZE=10;
        typedef long array[SIZE];
        void addArrays(in array a, in array b,
            out array result);
    };
};
```

- IDL übersetzen:

```
> idlj -fall Add.idl
```

[7] © Robert Tolksdorf, Berlin

Implementierung der Funktionalität

```
import ArithApp.*;
import org.omg.CORBA.*;

class AddImpl extends AddPOA {
    private ORB orb;

    public AddImpl(ORB orb) {
        this.ORB = orb;
    }

    // implement the addArrays() method
    public void addArrays(int a[], int b[],
        ArithApp.AddPackage.arrayHolder result) {

        result.value = new int[ArithApp.Add.SIZE];
        for(int i=0; i<ArithApp.Add.SIZE; i++) {
            result.value[i] = a[i] + b[i];
        }
    }
}
```

[8] © Robert Tolksdorf, Berlin

Implementierung des Serverobjekts

- Imports und Initialisierung des ORB

```
import ArithApp.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import org.omg.CosNaming.NamingContextPackage.*;

public class AddServer {

    public static void main(String args[]) {
        try{
            // create and initialize the ORB
            ORB orb = ORB.init(args, null);
```

[9] © Robert Tolksdorf, Berlin

Der ORB

[10] © Robert Tolksdorf, Berlin

Das ORB/CORBA Objekt

- Der ORB ist durch eine IDL Schnittstelle definiert

```
module CORBA {
    interface NVList; // forward declaration
    interface OperationDef; // forward declaration
    interface TypeCode; // forward declaration
    typedef short PolicyErrorCode;
    // for the definition of consts see "PolicyErrorCode" on page 4-39
    typedef unsigned long PolicyType;
    interface Request; // forward declaration
    typedef sequence <Request> RequestSeq;
    native AbstractBase;
    exception PolicyError {PolicyErrorCode reason;};
    typedef string RepositoryId;
    typedef string Identifier;
    // StructMemberSeq defined in Chapter 10
    // UnionMemberSeq defined in Chapter 10
    // EnumMemberSeq defined in Chapter 10
    typedef unsigned short ServiceType;
    typedef unsigned long ServiceOption;
    typedef unsigned long ServiceDetailType;
    const ServiceType Security = 1;
    ...
}
```

[11] © Robert Tolksdorf, Berlin

Java Mapping

- Durch Java-Binding/Mapping abgebildet auf die Klasse org.omg.CORBA.ORB
- API enthält
 - Basisdefinition zum externen IDL Typmanagement
 - Verwaltungsfunktionen
 - Arbeitsfunktionen
 - weitere
- Studium der Dokumentation unerlässlich
 - CORBA Spezifikation
Hier lokal als
\\nbi\agnbi\ag-nbi\lehre\NP\CORBA\CORBA.pdf
 - JDK Dokumentation
Als Javadoc beiliegend

[12] © Robert Tolksdorf, Berlin

ORB Initialisierung (Java)

- Methoden zur Initialisierung des Laufzeitsystems ORB
 - static ORB init()
 - static ORB init(Applet app, Properties props)
 - static ORB init(String[] args, Properties props)
- Kommandozeilenargumente haben die Form
-ORB<suffix> <optional white space> <value>
- Beispiele:
 - ORBNoProprietaryActivation
 - ORBInitRef NameService=IOR:00230021AB...
- Wichtig:
 - ORBInitialHost namerserverhost
Wo ist der Namensdienst (Normalfall: localhost)
 - ORBInitialPort nameserverport
Auf welchem Port arbeitet der Namensdienst (Normalfall: 900)
- Weitere Methoden, beispielsweise zur Termination

[13] © Robert Tolksdorf, Berlin

ORB Initialisierung (Java)

- Client kann auch selber die Angaben zum Namensdienst vorgeben:

```
Properties props = new Properties();
props.put("org.omg.CORBA.ORBInitialPort", "1050");
props.put("org.omg.CORBA.ORBInitialHost", "localhost");
ORB orb = ORB.init(args, props);
```

[14] © Robert Tolksdorf, Berlin

Verwaltung von Basisreferenzen

- Initiale Referenzen zum Auffinden von Basisdiensten
 - public abstract String[] list_initial_services()
 - Liefert eine Liste der initial bekannten Dienste zurück
 - Beim JDK ORB:

ServerActivator	ServerLocator
NameService	InitialNameService
ServerRepository	RootPOA
CodecFactory	DynAnyFactory
POACurrent	PICurrent
- Referenzen auf initiale Dienst liefern
 - abstract Object resolve_initial_references(String object_name)
 - Beispiel:

```
org.omg.CORBA.Object objRef =
orb.resolve_initial_references("NameService");
```

[15] © Robert Tolksdorf, Berlin

Bereitstellung externer Objektreferenzen

- Konvertierung zwischen (CORBA) Objektreferenzen und Zeichenketten (-> rmi: URLs)
 - public abstract String object_to_string(Object obj)
 - public abstract Object string_to_object(String str)
- Beispiel

```
org.omg.CORBA.Object naming=orb.resolve_initial_references("NameService");
String s=orb.object_to_string(naming);
System.out.println(s);
org.omg.CORBA.Object o=orb.string_to_object(s);
System.out.println(o.equals(naming));
```
- Ergebnis (IOR Schema ist portabel):

```
>java ORBInfo
IOR:000000000000002b49444c3a6f6d672e6f72672f436f734e616d696e672f4e616d696e674
36f6e746578744578743a312e30000000000010000000000008400010200000000f31
36302e34352e3131342e3230340000038400000000035afabcb0000000020a5f476f40000
000100000000000000010000000d544e616d65536572766963650000000000000040000
00000a00000000000001000000010000002000000000001000100000002050100010001
0020000101090000000100010100
```

true
- Sun Java ORB Implementierung: Zusätzlich corbaloc: und corbaname: URL Schemata

[16] © Robert Tolksdorf, Berlin

Implementierung des Serverobjekts

- Imports und Initialisierung des ORB

```
import ArithApp.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
import org.omg.CosNaming.NamingContextPackage.*;

public class AddServer {

    public static void main(String args[]) {
        try{
            // create and initialize the ORB
            ORB orb = ORB.init(args, null);
```

[17] © Robert Tolksdorf, Berlin

Implementierung des Serverobjekts/2

- Implementierung erzeugen und registrieren

```
// create an impl. and register it with the ORB
AddImpl impl = new AddImpl(orb);

// get ref. to rootpoa & activate the POAManager
POA rootpoa = POAHelper.narrow(
    orb.resolve_initial_references("RootPOA"));
rootpoa.the_POAManager().activate();

// get object reference from the servant
org.omg.CORBA.Object ref =
    rootpoa.servant_to_reference(impl);
Add href = AddHelper.narrow(ref);
```

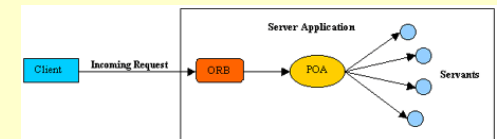
[18] © Robert Tolksdorf, Berlin

Der POA

[19] © Robert Tolksdorf, Berlin

POA

- Portable Object Adapter steht zwischen dem ORB und Anwendungsobjekten
- Er kann bestimmte Policies verfolgen
- Referenz auf obersten POA bekommen
- POA kann in den Zuständen active, inactive, holding, und discarding sein und kann Zustände wechseln
- Der POA Manager repräsentiert den Zustand
 - POAManager the_POAManager()
- Aktivieren des POA:



```
POA rootpoa = POAHelper.narrow(
    orb.resolve_initial_references("RootPOA"));
```

```
rootpoa.the_POAManager().activate();
```

[20] © Robert Tolksdorf, Berlin

Eigene POA Arten

- Man kann sich einen eigenen POA mit geänderten Policies erzeugen
 - POA create_POA(String name, POAManager a_POAMan, Policy[] pols)
 - Policies:
 - IdAssignmentPolicy
create_id_assignment_policy(IdAssignmentPolicyValue value)
 - IdUniquenessPolicy
create_id_uniqueness_policy(IdUniquenessPolicyValue value)
 - ImplicitActivationPolicy
create_implicit_activation_policy(ImplicitActivationPolicyValue val)
 - LifespanPolicy create_lifespan_policy(LifespanPolicyValue value)
 - RequestProcessingPolicy
create_request_processing_policy(RequestProcessingPolicyValue value)
 - ServantRetentionPolicy
create_servant_retention_policy(ServantRetentionPolicyValue val)
 - ThreadPolicy create_thread_policy(ThreadPolicyValue value)

[21] © Robert Tolksdorf, Berlin

POA Bäume und Referenzen

- POAs bilden einen Baum
 - POA the_parent()
 - POA[] the_children()
 - String the_name()
- Referenzen müssen Objekt hinter POA bezeichnen
- POA kann Referenzen erzeugen
 - Servant reference_to_servant(Object reference)
Object ist hier org.omg.CORBA.Object!
 - Object servant_to_reference(Servant p_servant)
- Diese Referenzen können registriert werden

[22] © Robert Tolksdorf, Berlin

Implementierung des Serverobjekts/2

- Implementierung erzeugen und registrieren

```
// create an impl. and register it with the ORB
AddImpl impl = new AddImpl(orb);

// get ref. to rootpoa & activate the POAManager
POA rootpoa = POAHelper.narrow(
    orb.resolve_initial_references("RootPOA"));
rootpoa.the_POAManager().activate();

// get object reference from the servant
org.omg.CORBA.Object ref =
    rootpoa.servant_to_reference(impl);
Add href = AddHelper.narrow(ref);
```

[23] © Robert Tolksdorf, Berlin

Implementierung des Serverobjekts/3

- Referenz unter Namen binden

```
// get the root naming context
// NameService invokes the name service
org.omg.CORBA.Object objRef =
    orb.resolve_initial_references("NameService");
// Use NamingContextExt, part of the Interoperable
// Naming Service (INS) specification.
NamingContextExt ncRef =
    NamingContextExtHelper.narrow(objRef);

// bind the Object Reference in Naming
String name = "Add";
NameComponent path[] = ncRef.to_name(name);
ncRef.rebind(path, href);
System.out.println("AddServer
    ready to add up your arrays ....");
```

[24] © Robert Tolksdorf, Berlin

Der NameService

[25] © Robert Tolksdorf, Berlin

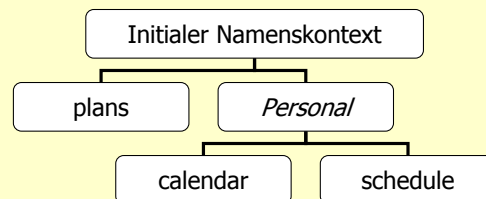
NameService API

- Der Namensdienst ist ein Standarddienst in CORBA
- Er ist definiert durch eine IDL und als OMG Standard festgelegt, siehe [\\Nbi\agnbi\ag-nbi\lehre\NP\CORBA\NamingService.pdf](http://Nbi\agnbi\ag-nbi\lehre\NP\CORBA\NamingService.pdf)
- Im JDK 1.4 sind die Java-Mappings des Dienstes im Paket `org.omg.CosNaming`
- Die Pakete `org.omg.CosNaming.NamingContextExtPackage` und `org.omg.CosNaming.NamingContextPackage` enthalten eine Verfeinerung des Namensdienstes von Sun (z.B. zusätzliche URL Schemata)
- Der Dienst hat den Namen **NameService**

[26] © Robert Tolksdorf, Berlin

Namensbindungen und -kontexte

- Zwischen Namen und Objekten können *Bindungen* erstellt werden
- Jede Bindung ist relativ zu einem *Namenskonzext*
- In einem Namenskontext sind alle gebundenen Namen eindeutig
- Ein Namenskontext selber kann auch an einen Namen gebunden werden
- Dadurch entsteht ein Namensgraph
- Das *Auflösen* eines Namens ist die Abfrage der gebundenen Referenz



[27] © Robert Tolksdorf, Berlin

Binden eines Namens

- Initialen Namenskontext abfragen

```
NamingContextExt ctx =
    NamingContextExtHelper.narrow(
        orb.resolve_initial_references("NameService"));
org.omg.CORBA.Object dummy = ctx;
```
- Namensobjekt erzeugen

```
NameComponent name1[] = ctx.to_name("plans");
```
- Dummy-Objekt an den Namen binden

```
ctx.rebind(name1, objref);
```

[28] © Robert Tolksdorf, Berlin

Neuen Kontext einfügen

- Neuen Kontext erzeugen und binden

```
NameComponent name2[] = ctx.to_name("Personal");
NamingContextExt ctx2 =
    (NamingContextExt)ctx.bind_new_context(name2);
```

- Weitere Bindungen einfügen

```
NameComponent name3[] = ctx.to_name("schedule");
ctx2.rebind(name3, objref);
NameComponent name4[] = ctx.to_name("calendar");
ctx2.rebind(name4, objref);
```

[29] © Robert Tolksdorf, Berlin

Bindung auflösen

- NameService erhalten und Namen auflösen

```
NamingContextExt nc =
    NamingContextExtHelper.narrow(
        orb.resolve_initial_references("NameService"));
org.omg.CORBA.Object sched =
    nc.resolve_str("Personal/schedule");
org.omg.CORBA.Object cal =
    nc.resolve_str("Personal/calendar");
org.omg.CORBA.Object plan =
    nc.resolve_str("plans");
```

[30] © Robert Tolksdorf, Berlin

Implementierung des Serverobjekts/3

- Referenz unter Namen binden

```
// get the root naming context
// NameService invokes the name service
org.omg.CORBA.Object objRef =
    orb.resolve_initial_references("NameService");
// Use NamingContextExt, part of the Interoperable
// Naming Service (INS) specification.
NamingContextExt ncRef =
    NamingContextExtHelper.narrow(objRef);

// bind the Object Reference in Naming
String name = "Add";
NameComponent path[] = ncRef.to_name(name);
ncRef.rebind(path, href);
System.out.println("AddServer
    ready to add up your arrays ....");
```

[31] © Robert Tolksdorf, Berlin

Implementierung des Serverobjekts/4

- Serverobjekt ausführen

```
// wait for invocations from clients
orb.run();
} catch (Exception e) {
    System.err.println("ERROR: " + e);
    e.printStackTrace(System.out);
}
System.out.println("AddServer Exiting ....");
}
```

[32] © Robert Tolksdorf, Berlin

Ausnahmen

Ausnahmen

- In CORBA signalisieren Ausnahmen Fehler
- Sie können bei Ausführung einer Methode auftreten
- Zwei Arten:
 - System Ausnahmen
 - Betreffen Ausnahmesituationen des Systems
 - Sind von OMG standardisiert
 - Können bei jeder Ausführung einer Methode auftreten
 - Anwender Ausnahmen (user exceptions)
 - Betreffen anwendungsspezifische Ausnahmesituationen
 - Werden vom Programmierer definiert
 - Können dort auftreten, wo sie mit **raises** in der IDL „angekündigt“ werden

Systemausnahmen

- Ausnahmequellen
 - Serverseitig (Ressourcenüberlastung, Serverobjektfehler)
 - Kommunikationsseitig (Netzausfall)
 - Klientenseitig (Argumentfehler)
- Ausnahmen haben immer den Typ
`{unsigned long minor; completion_status completed;}`
mit

```
enum completion_status {  
    COMPLETED_YES,  
    COMPLETED_NO,  
    COMPLETED_MAYBE  
};
```

Ausnahmen

- Der minor-Code ist ORB-herstellerabhängige zusätzliche Information
- Der Completion-status gibt an, wann die Ausnahme auftrat:
 - COMPLETED_YES: Nach Objektaufruf
 - COMPLETED_NO: Vor Objektaufruf
 - COMPLETED_MAYBE: unbekannt

Systemausnahmen

- exception UNKNOWN ex_body; // the unknown exception
- exception BAD_PARAM ex_body; // an invalid parameter was passed
- exception NO_MEMORY ex_body; // dynamic memory allocation failure
- exception IMP_LIMIT ex_body; // violated implementation limit
- exception COMM_FAILURE ex_body; // communication failure
- exception INV_OBJREF ex_body; // invalid object reference
- exception NO_PERMISSION ex_body; // no permission for attempted op.
- exception INTERNAL ex_body; // ORB internal error
- exception MARSHAL ex_body; // error marshaling param/result
- exception INITIALIZE ex_body; // ORB initialization failure
- exception NO_IMPLEMENT ex_body; // operation impl unavailable
- exception BAD_TYPECODE ex_body; // bad typecode
- exception BAD_OPERATION ex_body; // invalid operation
- exception NO_RESOURCES ex_body; // insufficient resources for req.
- exception NO_RESPONSE ex_body; // response to req. not yet available
- exception PERSIST_STORE ex_body; // persistent storage failure
- exception BAD_INV_ORDER ex_body; // routine invocations out of order
- exception TRANSIENT ex_body; // transient failure - reissue request
- exception FREE_MEM ex_body; // cannot free memory
- exception INV_IDENT ex_body; // invalid identifier syntax

[37] © Robert Tolksdorf, Berlin

Systemausnahmen

- exception INV_FLAG ex_body; // invalid flag was specified
- exception INTF_REPOS ex_body; // error accessing interface repository
- exception BAD_CONTEXT ex_body; // error processing context object
- exception OBJ_ADAPTER ex_body; // failure detected by object adapter
- exception DATA_CONVERSION ex_body; // data conversion error
- exception OBJECT_NOT_EXIST ex_body; // non-existent object, delete reference
- exception TRANSACTION_REQUIRED ex_body; // transaction required
- exception TRANSACTION_ROLLEDBACK ex_body; // transaction rolled back
- exception INVALID_TRANSACTION ex_body; // invalid transaction
- exception INV_POLICY ex_body; // invalid policy
- exception CODESET_INCOMPATIBLE ex_body; // incompatible code set
- exception REBIND ex_body; // rebind needed
- exception TIMEOUT ex_body; // operation timed out
- exception TRANSACTION_UNAVAILABLE ex_body; // no transaction
- exception TRANSACTION_MODE ex_body; // invalid transaction mode
- exception BAD_QOS ex_body; // bad quality of service
- exception INVALID_ACTIVITY ex_body; // bad quality of service
- exception ACTIVITY_COMPLETED ex_body; // bad quality of service
- exception ACTIVITY_REQUIRED ex_body; // bad quality of service

[38] © Robert Tolksdorf, Berlin

CORBA Ausnahmen in Java

- Systemausnahmen
 - sind Unterklassen von org.omg.CORBA.SystemException
 - sind damit Unterklassen von java.lang.RuntimeException
 - müssen also nicht behandelt werden
 - haben die Felder
 - CompletionStatus completed
 - int minor
- Anwendungsausnahmen
 - sind Unterklassen von org.omg.CORBA.UserException
 - sind damit Unterklassen von java.lang.Exception
 - müssen also immer behandelt werden

[39] © Robert Tolksdorf, Berlin

Beispiel

- Code aus [<http://developer.java.sun.com/developer/onlineTraining/corba/magercises/stock>]
- stock.idl

```
module StockObjects {
    struct Quote {
        string symbol; // The stock symbol of this quote.
        long long at_time; // Time when the price was updated.
        double price; // The current price of the stock.
        long volume; // The volume of shares traded.
    };

    exception Unknown{};
    interface Stock {
        Quote get_quote() raises(Unknown); // current quote
        readonly attribute string description; // description
    };
}; // end of module StockObjects
```

[40] © Robert Tolksdorf, Berlin

Beispiel

- StockImpl.java

```
import StockObjects.*;
public class StockImpl extends
    StockObjects._StockImplBase {
    private Quote _quote=null;
    private String _description=null;
    public StockImpl(String name,String description) {
        super();
        _description = description;
    }
    public Quote get_quote() throws Unknown {
        if (_quote==null) throw new Unknown();
        return _quote;
    }
    public String description() {
        return _description;
    }
}
```

[41] © Robert Tolksdorf, Berlin

Implementierung des Clients/1

- ORB intialisieren und Referenz auf Namenskontextholen

```
import ArithApp.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;

public class AddClient {
    public static void main(String args[]) {
        try {
            // create and initialize the ORB
            ORB orb = ORB.init(args, null);
            // get the root naming context
            org.omg.CORBA.Object objRef =
                orb.resolve_initial_references("NameService");
            // NamingContextExt instead of NamingContext. Is
            // part of the Interoperable Naming Service.
            NamingContextExt ncRef =
                NamingContextExtHelper.narrow(objRef);
```

[42] © Robert Tolksdorf, Berlin

Implementierung des Clients/2

- Referenz auf Server holen

```
// resolve the Object Reference in Naming
String name = "Add";
Add impl =
    AddHelper.narrow(ncRef.resolve_str(name));

System.out.println("Handle " +
    obtained on server object: " + impl);
```

[43] © Robert Tolksdorf, Berlin

Implementierung des Clients/3

- Parameter vorbereiten und Methode aufrufen

```
// the arrays to be added
int a[] = {6, 6, 6, 6, 6, 6, 6, 6, 6, 6};
int b[] = {7, 7, 7, 7, 7, 7, 7, 7, 7, 7};

// result is be saved in this new array
ArithApp.AddPackage.arrayHolder c =
    new ArithApp.AddPackage.arrayHolder();

// invoke the method addArrays()
impl.addArrays(a, b, c);
```

[44] © Robert Tolksdorf, Berlin

Implementierung des Clients/4

- Ausgeben und beenden

```
// print the new array
System.out.println("The sum of the "+
                   "two arrays is: ");
for(int i=0;i<ArithApp.Add.SIZE;i++) {
    System.out.println(c.value[i]);
}

} catch (Exception e) {
    System.out.println("ERROR : " + e) ;
    e.printStackTrace(System.out);
}
}
}
```

[45] © Robert Tolksdorf, Berlin

Übersetzen und Ausführen

- Übersetzen:
 - > javac *.java ArithApp/*.java
- Auf einer Maschine ausführen:
 - ORB Server starten:
 - > orbd -ORBInitialPort 2500
 - Addier-Server starten
 - > java AddServer -ORBInitialPort 2500
 - Addier-Client starten
 - > java AddClient -ORBInitialPort 2500

[46] © Robert Tolksdorf, Berlin

Verteilte Ausführung

- Auf zwei Maschinen ausführen:
 - ORB Server starten:
 - fock> orbd -ORBInitialPort 2500
 - Addier-Server starten
 - fock> java AddServer -ORBInitialPort 2500
 - Addier-Client starten
 - athos> java AddClient -ORBInitialPort 2500 -ORBInitialHost
fock.inf.fu-berlin.de
- Auf drei Maschinen ausführen:
 - ORB Server starten:
 - fock> orbd -ORBInitialPort 2500
 - Addier-Server starten
 - athos> java AddServer -ORBInitialPort 2500 -ORBInitialHost
fock.inf.fu-berlin.de
 - Addier-Client starten
 - nawab> java AddClient -ORBInitialPort 2500 -ORBInitialHost
fock.inf.fu-berlin.de

[47] © Robert Tolksdorf, Berlin

Persistentes Server Objekt

[48] © Robert Tolksdorf, Berlin

Persistentes Server Objekt

- Server-Objekt soll auch dann noch existieren, wenn der erzeugende Prozess (`java AddServer`) terminiert
- Implementierung unter anderem Namen:

```
import ArithApp.*;
import org.omg.CORBA.ORB;
class AddServant extends AddPOA {
    private ORB orb;
    public AddServant(ORB orb) {
        this.orb = orb;
    }
    // implement the addArrays() method
    public void addArrays(int a[], int b[],
        ArithApp.AddPackage.arrayHolder result) {
        result.value = new int[ArithApp.Add.SIZE];
        for(int i=0; i<ArithApp.Add.SIZE; i++) {
            result.value[i] = a[i] + b[i];
        }
    }
}
```

[49] © Robert Tolksdorf, Berlin

Persistentes Server Objekt

- Server Objekt zunächst wie bekannt:

```
import ArithApp.*;
import org.omg.CORBA.ORB;
import org.omg.CORBA.Object;
import org.omg.CosNaming.NameComponent;
import org.omg.CosNaming.NamingContextExt;
import org.omg.CosNaming.NamingContextExtHelper;
import org.omg.CORBA.Policy;
import org.omg.PortableServer.Servant;
import org.omg.PortableServer.*;
import org.omg.PortableServer.POA;
public class AddServer3 {
    public static void main(String args[]) {
        try {
            // create and initialize the ORB
            ORB orb = ORB.init(args, null);
            // create servant and instantiate it
            AddServant servant = new AddServant(orb);
            // get reference to rootpoa and activate the POAManager
            POA rootpoa = POAHelper.narrow(
                orb.resolve_initial_references("RootPOA"));
```

[50] © Robert Tolksdorf, Berlin

Persistentes Server Objekt

- Objekt an eigenen POA mit anderer Policy binden

```
// Create the Persistent Policy
Policy[] policy = new Policy[1];
policy[0] = rootpoa.create_lifespan_policy(
    LifespanPolicyValue.PERSISTENT);

// Create a persistent POA by passing the policy
POA poa =
    rootpoa.create_POA("childPOA", null, policy);
// Activate PersistentPOA's POAManager. Without
// this all calls to persistent server will hang
// because POAManager
// will be in the 'HOLD' state.
poa.the_POAManager().activate();
// Associate the servant with PersistentPOA
poa.activate_object( servant );
```

[51] © Robert Tolksdorf, Berlin

Persistentes Server Objekt

- Server registrieren

```
// Resolve RootNaming context and bind a name
// for the servant.
org.omg.CORBA.Object obj =
    orb.resolve_initial_references("NameService");
NamingContextExt rootContext =
    NamingContextExtHelper.narrow(obj);
//bind the object reference in the naming context
NameComponent[] nc =
    rootContext.to_name("AddServer");
rootContext.rebind(nc,
    poa.servant_to_reference(servant));
// wait for client requests
orb.run();
} catch (Exception e) {
    System.err.println("Exception
        in AddServer3 Startup " + e);
}
}
```

[52] © Robert Tolksdorf, Berlin

Client

```
import ArithApp.*;
import org.omg.CORBA.ORB;
import org.omg.CORBA.OBJ_ADAPTER;
import org.omg.CosNaming.NamingContext;
import org.omg.CosNaming.NamingContextHelper;
import org.omg.CosNaming.NameComponent;
import org.omg.PortableServer.POA;

public class AddClient3 {
    public static void main(String args[]) {
        try {
            // create and initialize the ORB
            ORB orb = ORB.init(args, null);
            org.omg.CORBA.Object obj = orb.string_to_object(
                "corbaname::localhost:2500#AddServer");
            Add impl = AddHelper.narrow(obj);
            // the arrays to be added
            int a[] = {6, 6, 6, 6, 6, 6, 6, 6, 6, 6};
            int b[] = {7, 7, 7, 7, 7, 7, 7, 7, 7, 7};
            // the result will be saved in this new array
            ArithApp.AddPackage.arrayHolder c =
                new ArithApp.AddPackage.arrayHolder();
```

[53] © Robert Tolksdorf, Berlin

Client

```
while(true) {
    System.out.println("Calling
        the persistent AddServer3..");
    impl.addArrays(a, b, c);
    // print the new array
    System.out.println("The sum of the two arrays is: ");
    for(int i=0;i<ArithApp.Add.SIZE;i++) {
        System.out.println(c.value[i]);
    }
    System.out.println("...will
        call the server again in a few seconds...");
    System.out.println("...if the
        server is down, it will be automatically restarted...");
    Thread.sleep(6000);
} catch ( Exception e ) {
    System.err.println( "Exception in AddClient3..." + e );
    e.printStackTrace( );
}
}
```

[54] © Robert Tolksdorf, Berlin

Starten

- Übersetzen:
 - > javac *.java ArithApp/*.java
- Auf einer Maschine ausführen:
 - ORB Server starten:
 - > orbd -ORBInitialPort 2500
 - Servertool zu Start des persistentes Objekts:
 - > servertool -ORBInitialPort 2500
 - Darin per Kommando anmelden
 - servertool > register -server AddServer3 -applicationName arithmetic -classpath C:\Persistent
 - Server registriert (serverid = 257)
- Addier-Client starten
 - > java AddClient -ORBInitialPort 2500

[55] © Robert Tolksdorf, Berlin

Servertool

- Mit Servertool kann Server heruntergefahren werden
 - servertool > shutdown -serverid 257
 - Server erfolgreich heruntergefahren
- Ein weiterer Aufruf von AddClient3 aktiviert ihn wieder
- Änderungen in der Aktivität des Serverobjekts werden auf diese Weise transparent für den Client

[56] © Robert Tolksdorf, Berlin

Zusammenfassung

Zusammenfassung

1. ORB
 1. Initialisieren
 2. POA erfragen
2. POA
 1. Aktivieren
 2. Mit Policies initialisieren
3. NameService
 1. Als Standarddienst auffindbar
 2. Namenskontexte und -bindungen
4. Ausführung
 1. Über Namensdienst verbunden
5. Persistente Serverobjekte
 1. Durch Policy Aktivierung transparent

Literatur

- **OMG. CORBA 3.0 Spezifikation.**
http://www.omg.org/technology/documents/formal/corba_2.htm