

D Überblick über die 9. Übung

- Namensdienst (CORBA Name Service)
- Portable Object Adapter (POA)
 - ◆ Policies
 - ◆ POA Schnittstelle
 - ◆ Persistente Referenzen

OOVS - Übung

D.1 Der Namensdienst

- Binden von Objektreferenzen an symbolische Namen (wie RMI-registry)
- Hierarchischer Namensraum

OOVS - Übung

1 IDL-Schnittstelle

```
module CosNaming { //PIDL

    typedef string Istring;
    struct NameComponent {
        Istring id;
        Istring kind;
    };
    typedef sequence <NameComponent> Name;

    interface NamingContext {
        void bind(in Name n, in Object obj)
            raises(NotFound, CannotProceed,
                InvalidName, AlreadyBound);
        void rebind(in Name n, in Object obj)
            raises(NotFound, CannotProceed, InvalidName);
        void bind_context(in Name n, in NamingContext nc)
            raises(NotFound, CannotProceed,
                InvalidName, AlreadyBound);

        // Fortsetzung folgt
    };
};
```

OOVS - Übung

1 IDL-Schnittstelle (Fortsetzung)

```
void rebind_context(in Name n, in NamingContext nc)
    raises(NotFound, CannotProceed, InvalidName);
void unbind(in Name n)
    raises(NotFound, CannotProceed, InvalidName);

Object resolve(in Name n)
    raises(NotFound, CannotProceed, InvalidName);

NamingContext new_context();
NamingContext bind_new_context(in Name n)
    raises(NotFound, AlreadyBound,
        CannotProceed, InvalidName);
void destroy() raises(NotEmpty);

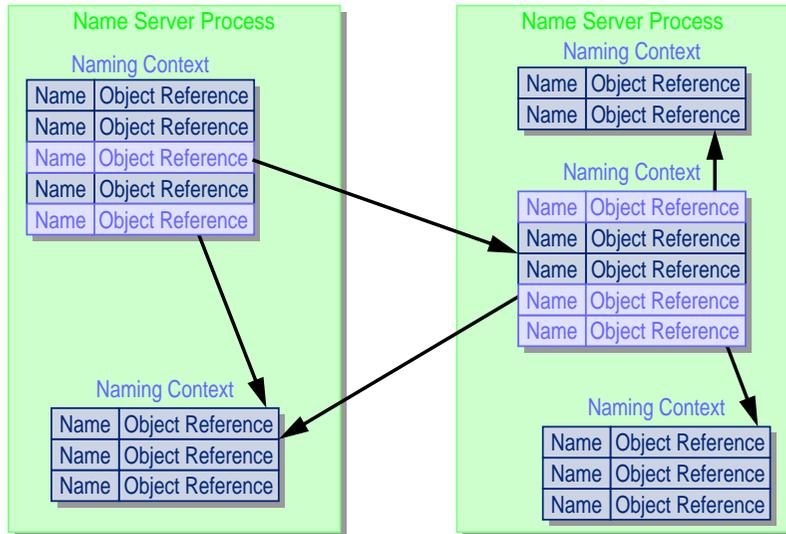
void list( in unsigned long    how_many,
          out BindingList      bl,
          out BindingIterator  bi);

    ...
}; // end of interface NamingContext
...
}; // end of module CosNaming
```

OOVS - Übung

2 Hierarchie von Namenskontexten

- Namenskontexte in mehreren Namensserver-Prozessen



4 "Hello World"-Client mit Namensdienst

- Bisher: ohne Namensdienst

```
//Ausschnitt aus der main-Methode
[...]
```

```
// Initialise ORB
ORB orb = ORB.init( args, null );
// Read object reference from file Hello.ior
BufferedReader br = new BufferedReader(
    new FileReader("Hello.ior"));

String s = br.readLine();
// Create a stub object
org.omg.CORBA.Object o = orb.string_to_object(s);
// Narrow to the Hello interface
Hello hello = HelloHelper.narrow( o );

[...]
```

3 Zugriff auf den "Root Naming Context"

- ORB kennt die Referenz

```
org.omg.CORBA.Object o =
    orb.resolve_initial_references("NameService");
org.omg.CosNaming.NamingContext root_context =
    org.omg.CosNaming.NamingContextHelper.narrow( o );
```

- ORB bekommt den Root Naming Context via Kommandozeilenparameter oder Properties

- ◆ Seit der "Interoperable Naming Service (INS)"-Spezifikation (Nov 2000)

- ◆ ORB wertet alle Parameter der Form `form -ORB<suffix> <value>`

- ◆ Für initiale Referenzen:

```
-ORBInitRef ObjectID=Reference
```

- ◆ Lesbare Objektreferenzen:

```
corbaloc:Protocol:Host:Port/ObjectId
```

- ◆ Beispiel:

```
-ORBInitRef NameService=corbaloc::fau140u.informatik.uni-erlangen.de:4711/NameService
```

4 "Hello World"-Client mit Namensdienst

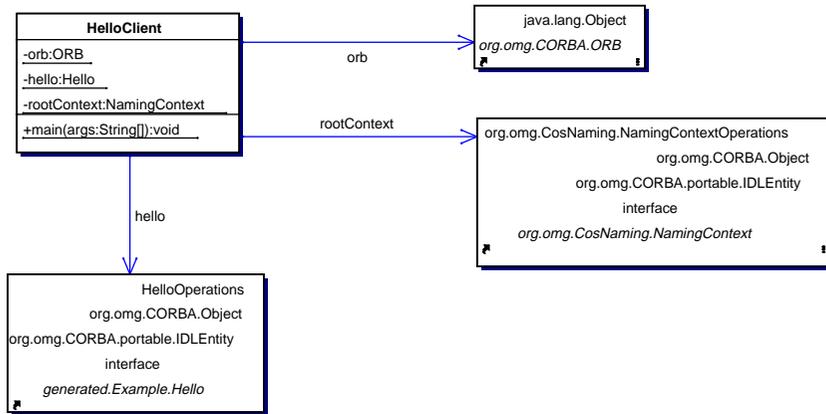
- Importieren eines zusätzlichen package

```
import org.omg.CosNaming.*;
```

- Änderungen an der main-Methode des Clients

```
// Initialise ORB
ORB orb = ORB.init( args, null );
// Get Name Service reference
org.omg.CORBA.Object objRootContext =
    orb.resolve_initial_references("NameService");
// Narrow it to the NamingContext interface
NamingContext rootContext =
    NamingContextHelper.narrow( objRootContext );
// Create a name as an array of NameComponent objects
NameComponent name[] = new NameComponent[2];
name[0] = new NameComponent("pub", "");
name[1] = new NameComponent("Hello", "");
// Look the object up
org.omg.CORBA.Object o = rootContext.resolve(name);
// Narrow to the Hello interface
Hello h = HelloHelper.narrow( o );
```

4 "Hello World"-Client mit Namensdienst (2)



4 "Hello World"-Server - ohne Namensdienst

```

// server/HelloServer.java
import generated.Example.*;
import org.omg.CORBA.*;
import java.io.*;
import org.omg.PortableServer.*;

public class HelloServer {
    public static void main( String[] args ) {
        try {
            // Initialize ORB
            ORB orb = ORB.init( args, null );

            // Get the RootPOA
            POA poa = POAHelper.narrow(
                orb.resolve_initial_references( "RootPOA" ) );

            // Activate the RootPOA
            poa.the_POAManager().activate();

            // Create the hello servant object
            HelloServant hServ = new HelloServant();

            // Activate the object
            Hello h = hServ._this( orb );

            // Fortsetzung folgt
        }
    }
}
  
```

4 "Hello World"-Client mit Namensdienst (3)

- In das Beispiel-Verzeichnis wechseln

```
> cd /proj/i4oovs/pub/hello_nameservice
```

- Compilieren

```
> idlj Hello.idl [JDK 1.4]
```

oder

```
> idl Hello.idl [JacORB 1.4]
```

```
> javac HelloClient.java [immer]
```

- Starten

```
> java HelloClient -ORBInitRef NameService=\
    corbaloc::fau140u.informatik.uni-erlangen.de:4711/\
    NameService [JDK 1.4]
```

bzw.

```
> jaco ... [JacORB 1.4]
```

4 "Hello World"-Server - ohne Namensdienst (2)

```

// Write reference
PrintWriter pw = new PrintWriter(new FileWriter(
    "/proj/i4oovs/pub/hello/Hello.ior"));
pw.println( orb.object_to_string( h ) );
pw.close();

// Wait for request
orb.run();

} catch( org.omg.CORBA.SystemException e ) {
    //...
} catch( Throwable t ) {
    //...
}
} // end of method main
} // end of class HelloServer
  
```

4 "Hello World"-Server mit Namensdienst

```
// server/HelloServer.java
import generated.Example.*;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.PortableServer.*;

public class HelloServer {
    public static void main( String[] args ) {
        try {
            // Initialize ORB
            ORB orb = ORB.init( args, null );

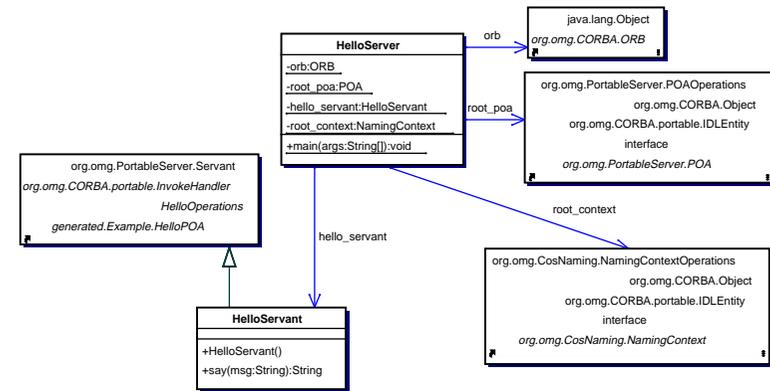
            // Get the RootPOA
            POA poa = POAHelper.narrow(
                orb.resolve_initial_references( "RootPOA" ) );

            // Activate the RootPOA
            poa.the_POAManager().activate();

            // Create the hello servant object
            HelloServant hServ = new HelloServant();

            // Activate the object
            Hello h = hServ._this( orb );
        } // Fortsetzung folgt
    }
}
```

4 "Hello World"-Server mit Namensdienst



4 "Hello World"-Server mit Namensdienst

```
// Get Name Service reference
org.omg.CORBA.Object obj_root_context =
    orb.resolve_initial_references("NameService");
// Narrow it to the NamingContext interface
root_context = NamingContextHelper.narrow(
    obj_root_context );
// Create a name as an array of NameComponents
NameComponent name[] = new NameComponent[2];
name[0] = new NameComponent("pub", "");
name[1] = new NameComponent("Hello", "");
// Register the Hello object
root_context.rebind(name, hello);

// Wait for request
orb.run();

} catch( org.omg.CORBA.SystemException e ) {
    //...
} catch( Throwable t ) {
    //...
}
} // end of method main
} // end of class HelloServer
```

4 "Hello World"-Server mit Namensdienst

■ Beispielverzeichnis:

```
> cd /proj/i4oovs/pub/hello_nameservice
```

■ Kompilieren

```
> idlj -fall Hello.idl
> javac HelloServer.java
```

■ Starten

```
> java HelloServer -ORBInitRef NameService=\
corbaloc::fau140u.informatik.uni-erlangen.de:4711/\
NameService
```

5 Zusammenfassung

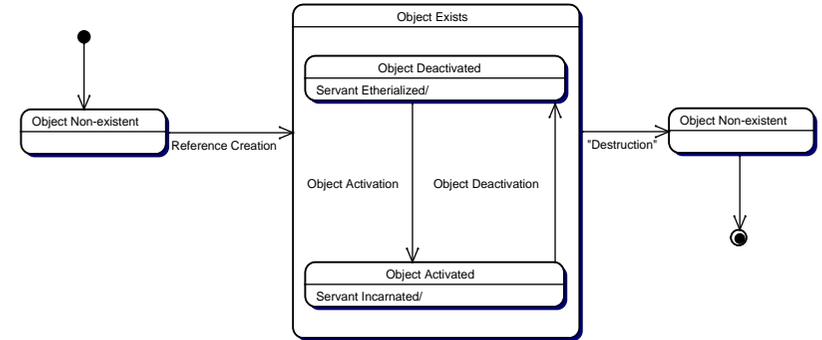
- Binden von Objektreferenzen an Namen
- "Naming Contexts" sind normale CORBA-Objekte (IDL-Interface)
- "Naming Contexts" residieren in Namensdienst-Prozessen
- Konfiguration des "Root Naming Context" für eine Anwendung über Kommandozeilenparameter

D.2 Portable Object Adaptor (POA)

- Aktivierungsschemas
- POA-Schnittstelle
- Persistente Referenzen

1 CORBA-Objekt-Lebenszyklus

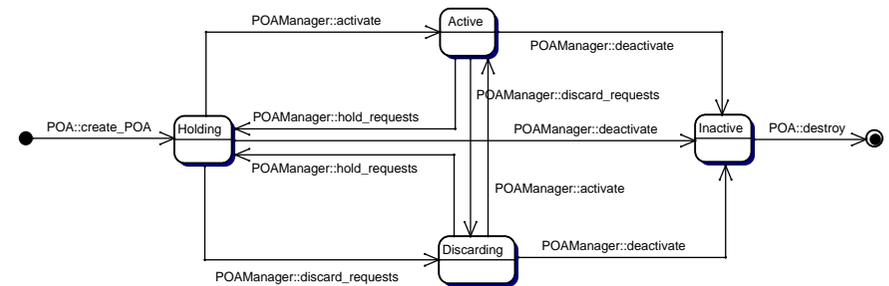
- Zustandsdiagramm für den Lebenszyklus



- ◆ OA muss bei Ankunft eines Anforderung den Servant erzeugen und das Objekt aktivieren

2 POA-Manager

- Der Betrieb des POA wird vom "POA-Manager" gesteuert
- Zustände des POA-Manager



- Ein POA-Manager für mehrere POAs möglich
- PIDL-Schnittstelle **POAManager**

3 POA-Erzeugung

■ IDL-Schnittstelle:

```

module PortableServer {                               //PIDL
  interface POAManager;
  exception AdapterAlreadyExists {};
  exception InvalidPolicy { unsigned short index; };

  interface POA {
    POA create_POA(in string          adapter_name,
                  in POAManager      manager,
                  in CORBA::PolicyList policies )
      raises(AdapterAlreadyExists, InvalidPolicy);
    ...
  };
};

```

■ Policies beeinflussen die Arbeitsweise des POA

■ Ein Root-POA existiert bereits im ORB

```

org.omg.CORBA.Object o =
  orb.resolve_initial_references("RootPOA" );
org.omg.PortableServer.POA root_poa =
  org.omg.PortableServer.POAHelper.narrow( o );

```

4 POA-Policies

■ IDL:

```

module CORBA {
  typedef unsigned long PolicyType;

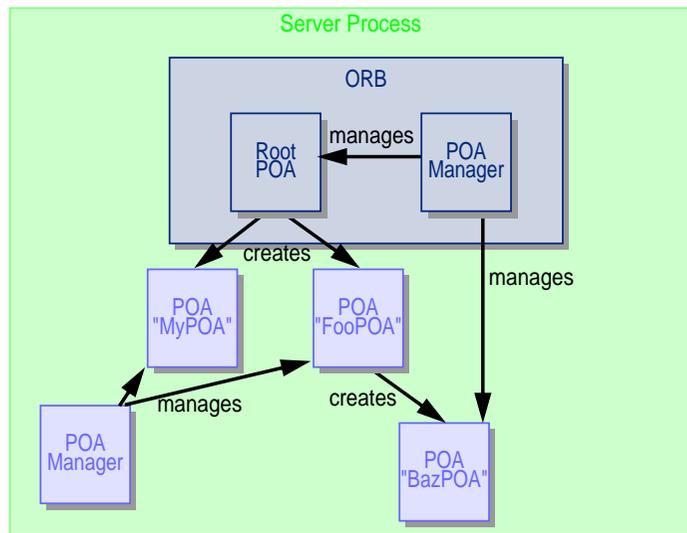
  interface Policy {
    readonly attribute PolicyType policy_type;

    Policy copy();
    void destroy();
  };
  typedef sequence<Policy> PolicyList;
};

```

3 POA-Erzeugung

■ Hierarchie



5 Lebensdauer-Policy (Lifespan)

■ IDL:

```

module PortableServer {
  enum LifespanPolicyValue {
    TRANSIENT, PERSISTENT
  };
  interface LifespanPolicy : CORBA::Policy {
    readonly attribute LifespanPolicyValue value;
  };
};

```

■ Ein einzelner POA kann entweder persistente oder transiente Objekte unterstützen, nicht beides

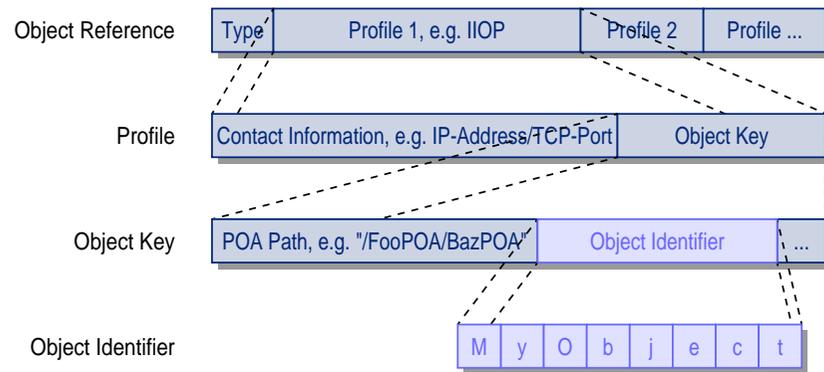
■ Persistente Objekte

- ◆ ORB und Implementation Repository müssen die Objekte verfolgen
- ◆ Zusätzliche Informationen für die Reaktivierung wird gespeichert

■ Default-Wert: **TRANSIENT**

6 Objekt-Identifikatoren

- Objekte werden über Objektreferenzen identifiziert



6 Objekt-Identifikatoren (2)

- Objekt-IDs werden festgelegt durch:
 - den POA (**SYSTEM_ID**)
 - die Anwendung (**USER_ID**), z.B. wenn Objekte auf Datenbank abgebildet werden

- IDL:

```
module PortableServer {
    enum IdAssignmentPolicyValue {
        SYSTEM_ID, USER_ID
    };
    interface IdAssignmentPolicy : CORBA::Policy {
        readonly attribute IdAssignmentPolicyValue value;
    };
};
```

- Default-Wert: **SYSTEM_ID**

7 Abbildung von Objekten zu Servants

- Beziehung zwischen Objekt-IDs und Servants
 - UNIQUE_ID:** Eins-zu-eins-Beziehung zwischen Objekten und Servants
 - MULTIPLE_ID:** Es kann mehrere CORBA-Objekte (und daher mehrere Objekt-IDs) geben, die durch den selben Servant implementiert werden

- IDL:

```
module PortableServer {
    enum IdUniquenessPolicyValue {
        UNIQUE_ID, MULTIPLE_ID
    };
    interface IdUniquenessPolicy : CORBA::Policy {
        readonly attribute IdUniquenessPolicyValue value;
    };
};
```

- Default-Wert: **UNIQUE_ID**

8 Implizite Aktivierung

- Aktivierung über eine spezielle Skeleton-Methode (in Java: **_this()**)

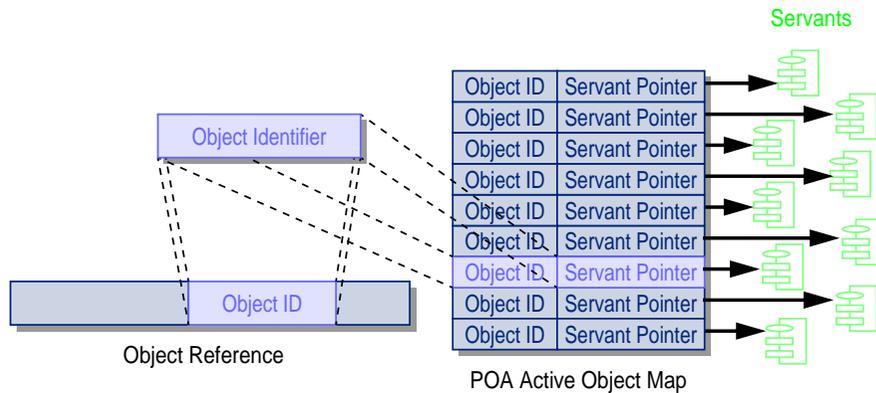
- IDL:

```
module PortableServer {
    enum ImplicitActivationPolicyValue {
        IMPLICIT_ACTIVATION, NO_IMPLICIT_ACTIVATION
    };
    interface ImplicitActivationPolicy : CORBA::Policy {
        readonly attribute ImplicitActivationPolicyValue value;
    };
};
```

- Default-Wert: **NO_IMPLICIT_ACTIVATION**
- Ausnahme: RootPOA verwendet **IMPLICIT_ACTIVATION**

9 Abbildung von Requests auf Servants

- POA kann aktive Servants in einem *Active Object Map* speichern



- Nur Objekte, die in der Active Object Map registriert sind, existieren
- POA erzeugt sonst `CORBA::OBJECT_NOT_EXIST`-Exception
- Policy: `USE_ACTIVE_OBJECT_MAP_ONLY`

9 Abbildung von Requests auf Servants

- IDL:

```
module PortableServer {
    enum RequestProcessingPolicyValue {
        USE_ACTIVE_OBJECT_MAP_ONLY,
        USE_DEFAULT_SERVANT,
        USE_SERVANT_MANAGER
    };
    interface RequestProcessingPolicy : CORBA::Policy {
        readonly attribute RequestProcessingPolicyValue
            value;
    };
};
```

- Default-Wert: `USE_ACTIVE_OBJECT_MAP_ONLY`

9 Abbildung von Requests auf Servants

- Für dynamische Aktivierung stellt die Anwendung dem POA einen *Servant-Manager* zur Verfügung
 - Servant-Manager wird befragt, falls die Objekt-ID nicht in der Active Object Map gefunden wird
 - Servant-Manager kann entweder einen Servant für das Objekt liefern, oder eine Exception erzeugen (`CORBA::OBJECT_NOT_EXIST`)
 - Policy: `USE_SERVANT_MANAGER`
- Die Abbildung stellt einen Default-Servant – es gibt keine Active Object Map
 - Alle Anforderungen werden an den Default-Servant geschickt
 - Dynamic Skeleton Interface (DSI) wird zur Bearbeitung der Anforderungen verwendet
 - Policy: `USE_DEFAULT_SERVANT`

10 Speicherung der Beziehung Objekt-ID zu Servant

- Soll der POA Servants in der Active Object Map speichern?
- Ja (**RETAIN**): POA durchsucht Active Object Map
- Nein (**NON_RETAIN**): POA verlässt sich auf Default-Servant oder Servant-Manager, um eine Zuordnung zu erhalten

- IDL:

```
module PortableServer {
    enum ServantRetentionPolicyValue {
        RETAIN, NON_RETAIN
    };
    interface ServantRetentionPolicy : CORBA::Policy {
        readonly attribute ServantRetentionPolicyValue
            value;
    };
};
```

- Default-Wert: `RETAIN`

11 Multithreading

- Wie werden Requests zu Threads zugeordnet?

- IDL:

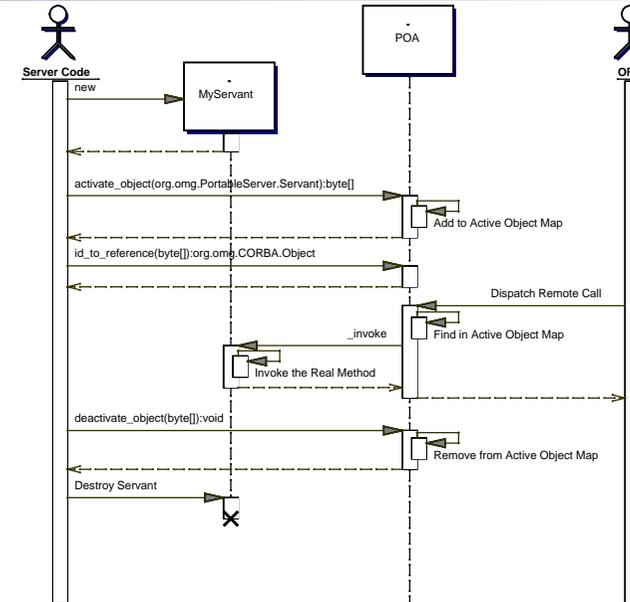
```
module PortableServer {
  enum ThreadPolicyValue {
    ORB_CTRL_MODEL, SINGLE_THREAD_MODEL
  };
  interface ThreadPolicy : CORBA::Policy {
    readonly attribute ThreadPolicyValue value;
  };
};
```

- Default-Wert: ORB_CTRL_MODEL

12 Nützliche Kombinationen von Policies

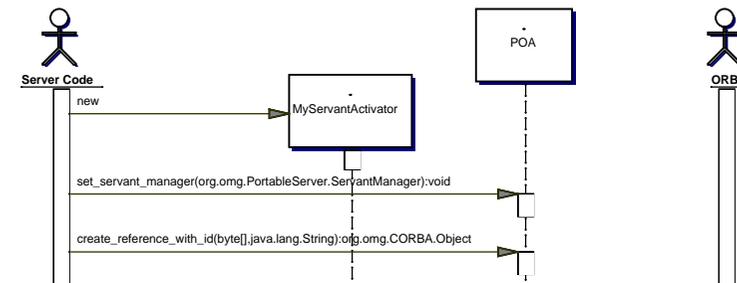
- Manche Kombinationen machen keinen Sinn, in diesen Fällen schlägt die POA-Erzeugung fehl.
- PERSISTENT** wird oft zusammen mit **USER_ID** verwendet
 - Servant kann leichter wiederhergestellt werden, wenn die Objekt-ID einen Schlüssel zum Finden der Servant-Daten enthält
- IMPLICIT_ACTIVATION** erfordert **SYSTEM_ID**
 - Wo soll die Objekt-ID herkommen?
- NON_RETAIN** nicht mit **USE_ACTIVE_OBJECT_MAP_ONLY**
 - Welcher Servant soll verwendet werden?

13 Einfacher POA ohne implizite Aktivierung

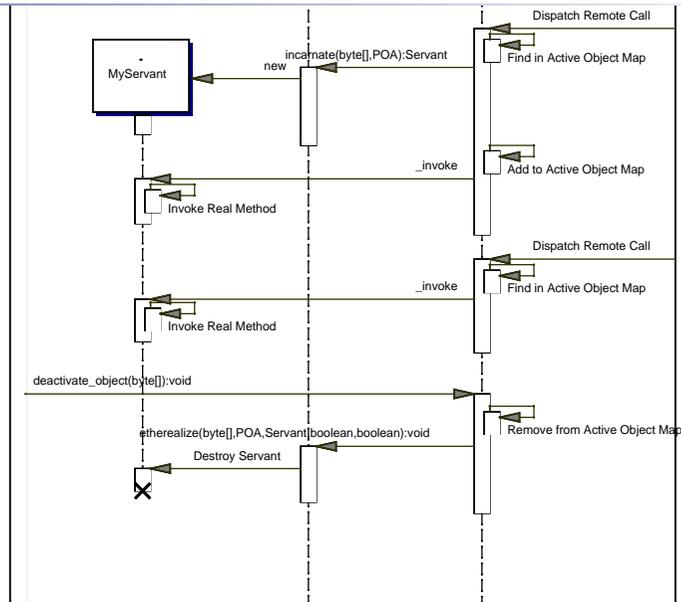


14 POA mit Servant-Activator

- Policies:
 - USE_SERVANT_MANAGER**
 - RETAIN**
- Anwendung aktiviert Objekte nach Bedarf via Servant-Manager
- Anwendung kann Objekte nach ihrer eigenen Strategie deaktivieren

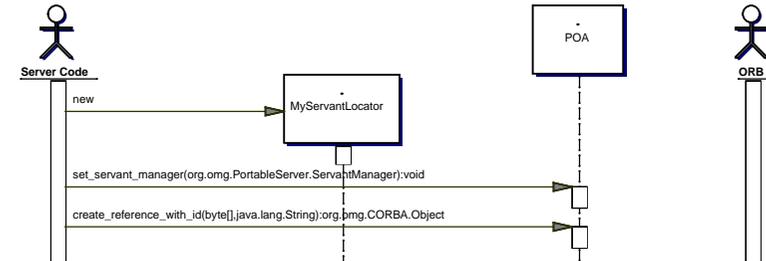


14 POA mit Servant-Activator



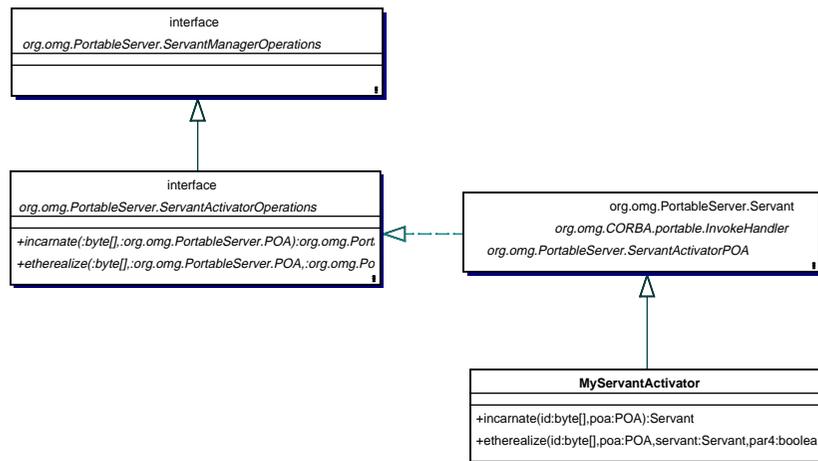
15 POA mit Servant-Locator

- Policies:
 - ◆ USE_SERVANT_MANAGER
 - ◆ NON_RETAIN
- POA hat keine "Active Object Map"
- Servant-Locator wird vor und nach jedem Request befragt
- Servant-Locator implementiert eigene Active Object Map

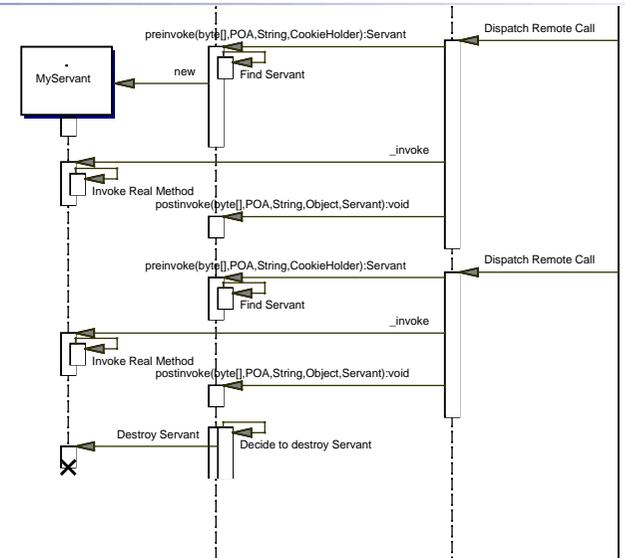


14 POA mit Servant-Activator

- Servant-Manager ist ein *Servant-Activator* (abgeleitete Schnittstelle)

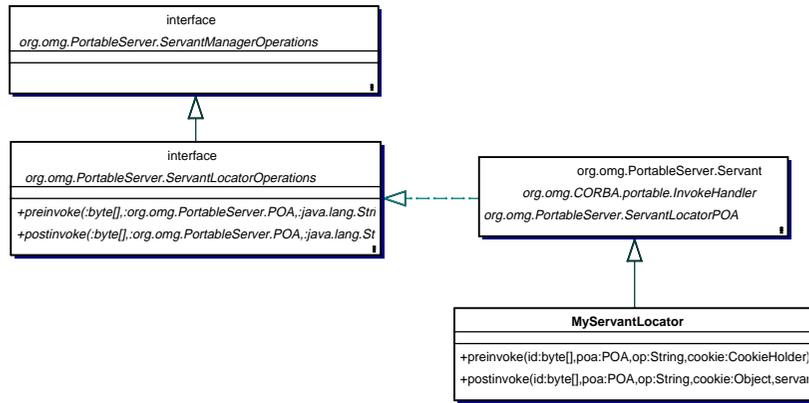


15 POA mit Servant-Locator



15 POA mit Servant-Locator

- Servant-Manager ist ein *Servant-Locator* (abgeleitete Schnittstelle)



16 POA mit Default-Servant

- Policies:
 - ◆ USE_DEFAULT_SERVANT
 - ◆ NON_RETAIN
- POA hat keine Active Object Map
- Anwendung stellt Default-Servant zur Verfügung
- Default-Servant bekommt vom POA jeden Request und verarbeitet ihn mit Hilfe des Dynamic Skeleton Interface (DSI)

17 POA-Schnittstelle

- POA-Attribute

```

readonly attribute string the_name;
readonly attribute POA the_parent;
readonly attribute POAList the_children;
readonly attribute POAManager the_POAManager;
attribute AdapterActivator the_aktivator;
  
```

- POA-Erzeugung

```

POA create_POA(in string adapter_name,
               in POAManager a_POAManager,
               in CORBA::PolicyList policies)
  raises (AdapterAlreadyExists, InvalidPolicy);
POA find_POA( in string adapter_name,
              in boolean activate_it)
  raises (AdapterNonExistent);
void destroy( in boolean etherealize_objects,
              in boolean wait_for_completion);
  
```

17 POA-Schnittstelle (2)

- Policy factory-Operationen

```

LifespanPolicy create_lifespan_policy(
  in LifespanPolicyValue value);
IdAssignmentPolicy create_id_assignment_policy(
  in IdAssignmentPolicyValue value);
IdUniquenessPolicy create_id_uniqueness_policy(
  in IdUniquenessPolicyValue value);
ImplicitActivationPolicy create_implicit_activation_policy(
  in ImplicitActivationPolicyValue value);
RequestProcessingPolicy create_request_processing_policy(
  in RequestProcessingPolicyValue value);
ServantRetentionPolicy create_servant_retention_policy(
  in ServantRetentionPolicyValue value);
ThreadPolicy create_thread_policy(
  in ThreadPolicyValue value);
  
```

17 POA-Schnittstelle (3)

■ Servant-Manager-Operationen

```
ServantManager get_servant_manager() raises (WrongPolicy);
void set_servant_manager(in ServantManager imgr)
    raises (WrongPolicy);
```

■ Default-Servant-Operationen

```
Servant get_servant() raises (NoServant, WrongPolicy);
void set_servant(in Servant p_servant)
    raises (WrongPolicy);
```

■ Operationen zur Objektaktivierung und -deaktivierung

```
ObjectId activate_object(in Servant p_servant)
    raises (ServantAlreadyActive, WrongPolicy);
void activate_object_with_id(in ObjectId id,
    in Servant p_servant)
    raises (ServantAlreadyActive,
    ObjectAlreadyActive, WrongPolicy);
void deactivate_object(in ObjectId oid)
    raises (ObjectNotActive, WrongPolicy);
```

17 POA-Schnittstelle (4)

■ Operationen zur Abbildung der ID

```
ObjectId servant_to_id(in Servant p_servant)
    raises (ServantNotActive, WrongPolicy);
Object servant_to_reference(in Servant p_servant)
    raises (ServantNotActive, WrongPolicy);

Servant reference_to_servant(in Object reference)
    raises (ObjectNotActive, WrongPolicy);
ObjectId reference_to_id(in Object reference)
    raises (WrongAdapter, WrongPolicy);

Servant id_to_servant(in ObjectId oid)
    raises (ObjectNotActive, WrongPolicy);
Object id_to_reference(in ObjectId oid)
    raises (ObjectNotActive, WrongPolicy);
```

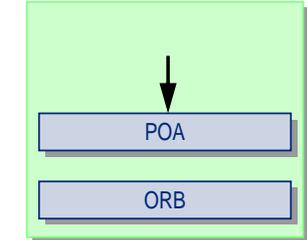
■ Operationen zur Erzeugung von Referenzen

```
Object create_reference(in CORBA::RepositoryId intf)
    raises (WrongPolicy);
Object create_reference_with_id(in ObjectId oid,
    in CORBA::RepositoryId intf)
    raises (WrongPolicy);
```

18 Persistente Referenzen

- Server-Prozess startet und fordert beim POA die Erzeugung einer Referenz an

Server Process at host2:1234

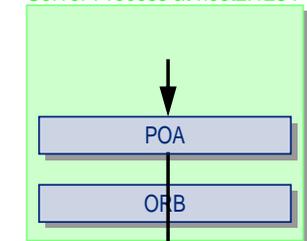


Implementation
Repository
at host2:4711

18 Persistente Referenzen

- Persistenter POA registriert Server im Implementation Repository (IR)

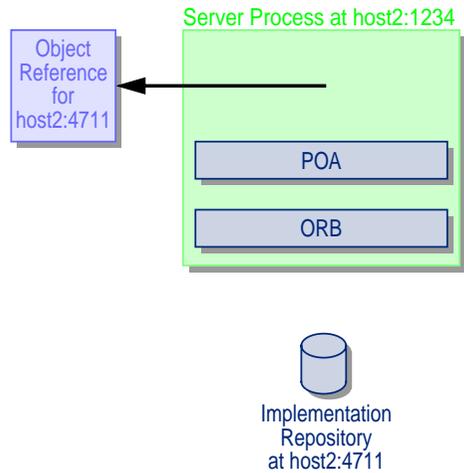
Server Process at host2:1234



Implementation
Repository
at host2:4711

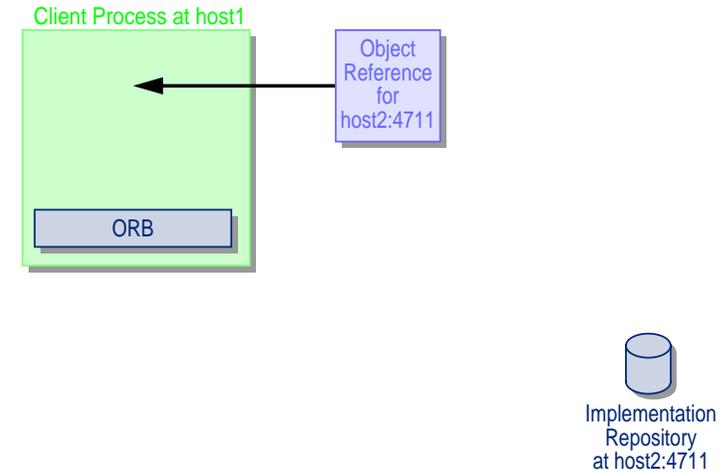
18 Persistente Referenzen

- Server speichert die vom POA erzeugte persistente Objektreferenz



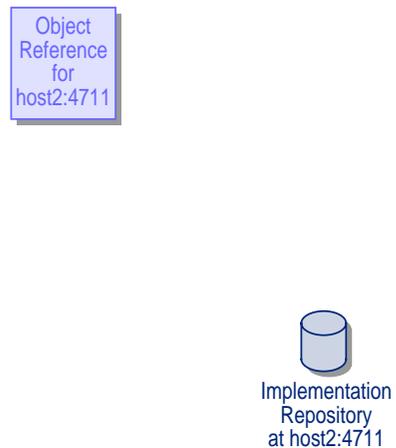
18 Persistente Referenzen

- Client wird gestartet und liest die Objekt-Referenz



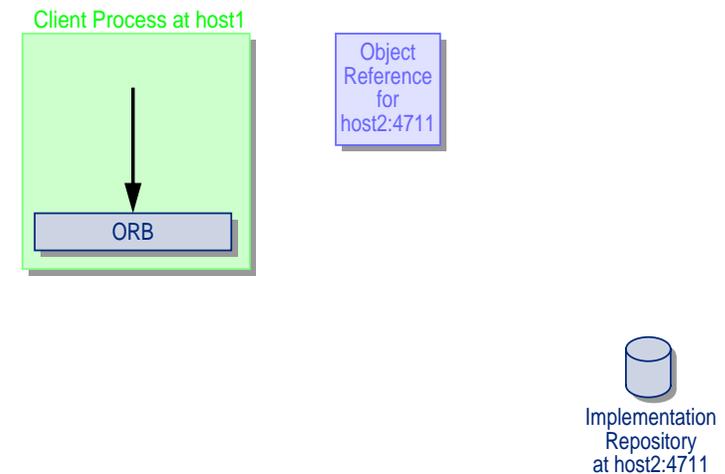
18 Persistente Referenzen

- Server wird beendet



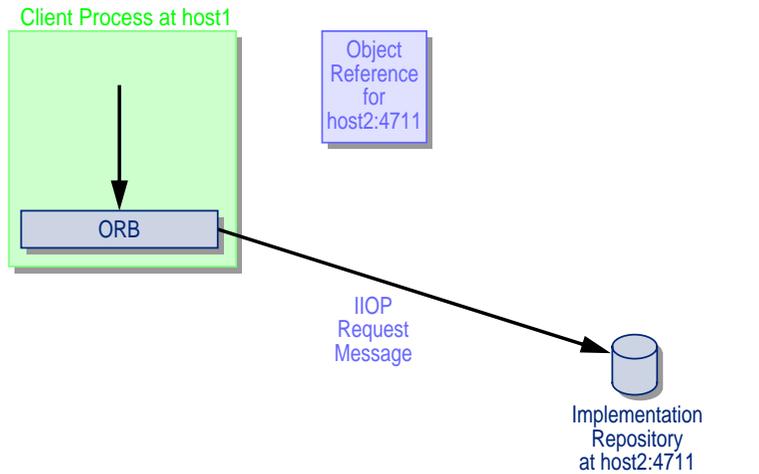
18 Persistente Referenzen

- Client ruft eine Operation auf



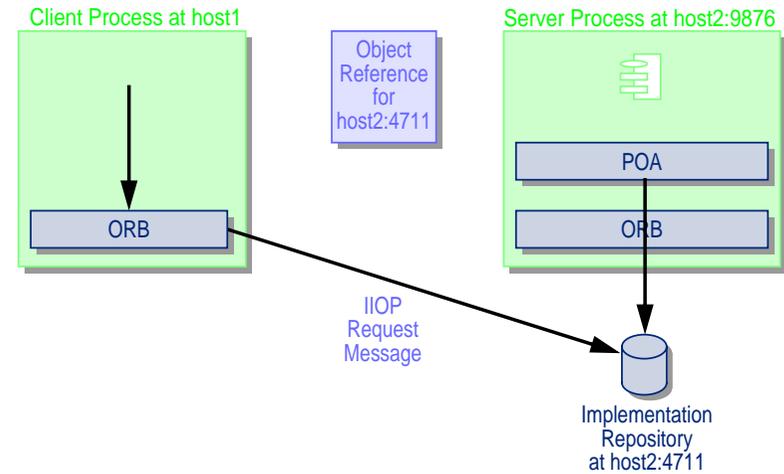
18 Persistente Referenzen

- Aufruf-Anforderung wird an die Kontaktadresse (d.h. IR) gesendet



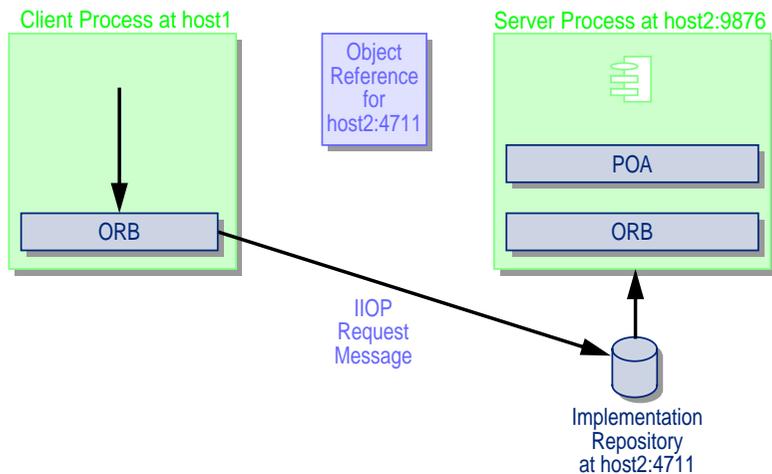
18 Persistente Referenzen

- POA registriert nun neue Kontakt-Adresse im IR



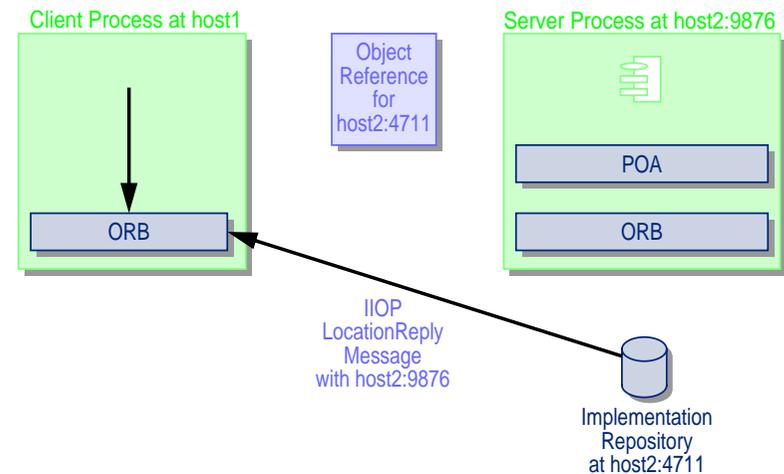
18 Persistente Referenzen

- IR startet Server-Prozess



18 Persistente Referenzen

- IR liefert "location forward"-Nachricht mit neuer Kontaktadresse



OOVS - Übung

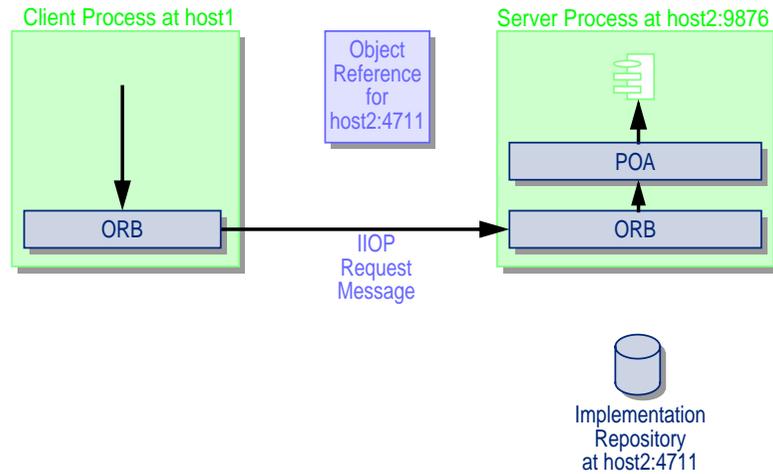
OOVS - Übung

OOVS - Übung

OOVS - Übung

18 Persistente Referenzen

- Aufruf wird an die neue Kontaktadresse gesendet und dort ausgeführt



19 Zusammenfassung POA

- Hierarchie aus POAs
- Viele verschiedene Policies
- Viele Arten von Anfrage-Bearbeitung und Servant-Verwaltungsstrategien sind möglich
- Persistente Referenzen via Implementation Repository

18 Persistente Referenzen

- Das Ergebnis wird zum Client zurückgeschickt

