B Überblick über die 10. Übung

- Besprechung der Aufgabe 5
- Remote Events
- ServiceEvents
- Security-Policies
- RMI Versionen

B.1 Lösung der Aufgabe 5

- library.server.Main (1/4)

```
package library.server;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.PortableServer.*;
import library.*;
import library.database.ItemDatabase;
import java.io.*;

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

            // Initialize Item Database
            ItemDatabase db = new ItemDatabase();

            // Find the root POA & start it
            POA rootPOA = POAHelper.narrow(
                orb.resolve_initial_references("RootPOA");
            rootPOA.the_POAManager().activate();
```


B.1 Lösung der Aufgabe 5

- library.server.Main (2/4)

```
// Set policies for POA that uses servant locator
Policy[] policies = new Policy[] {
    rootPOA.create_id_assignment_policy(
        IdAssignmentPolicyValue.USER_ID),
    rootPOA.create_servant_retention_policy(
        ServantRetentionPolicyValue.NON_RETAIN),
    rootPOA.create_request_processing_policy(
        RequestProcessingPolicyValue.USE_SERVANT_MANAGER)
};
POA itemPOA = rootPOA.create_POA("itemPOA",
    rootPOA.the_POAManager(), policies);

org.omg.PortableServer.ServantLocator locator =
    new ItemServantLocator(orb, db);
itemPOA.set_servant_manager(locator);

// Create a LibraryDB
LibraryDBServant libServ = new LibraryDBServant(
    rootPOA, itemPOA, db);
LibraryDB lib = LibraryDBHelper.narrow(
    rootPOA.servant_to_reference(libServ));
// deprecated: lib = libServ._this(orb)
// bzw: poa.id_to_reference(poa.activate_object(.))
```


B.1 Lösung der Aufgabe 5

- library.server.Main (3/4)

```
// Register at name service
// Read the reference
BufferedReader br = new BufferedReader( new FileReader(
    "/proj/i4mw/pub/aufgabe5/Nameservice.ior" ) );
String ior = br.readLine();

// Create a stub object
org.omg.CORBA.Object obj_root_context =
    orb.string_to_object( ior );

// Get name service reference
// org.omg.CORBA.Object obj_root_context =
//    orb.resolve_initial_references("NameService");

NamingContext root_context =
    NamingContextHelper.narrow(obj_root_context);
```

B.1 Lösung der Aufgabe 5

- library.server.Main (4/4)

```
    // Register new Namingcontext
    try {
        root_context.bind_new_context( new NameComponent[]
            { new NameComponent("mw", "")});
        } catch (AlreadyBoundException e)
        { System.err.println("   already registered");}

       // Create a name as an array of NameComponents
        NameComponent name[] = new NameComponent[2];
        name[0] = new NameComponent("mw", "");
        name[1] = new NameComponent("LibraryDB", "");
        // Register the Library object
        root_context.rebind(name, lib);

        System.out.println( "server up and running." );
        // Wait for requests
        orb.run();
    } catch( Exception ex ) { ex.printStackTrace(); }
}
```


B.1 Lösung der Aufgabe 5

- library.server.LibraryDBServant (1/3)

```
package library.server;

import library.*;
import library.database.*;
import org.omg.CORBA.*;
import org.omg.PortableServer.*;

public class LibraryDBServant extends LibraryDBPOA
{
    private POA my_poa;
    private ItemDatabase my_db;
    private POA item_poa;
    private ORB my_orb;

    public LibraryDBServant( ... ) { ... }
```


B.1 Lösung der Aufgabe 5

- library.server.LibraryDBServant (2/3)

```
public void register(String title)
    throws AlreadyExistsException;
{
    try {
        get(title);
        throw new AlreadyExistsException();
    } catch(NotFoundException e) {
        Record rec = my_db.newRecord();
        rec.setData("TITLE", title);
        rec.setData("BORROWED", "false");
        rec.setData("LENDINGCOUNT", "0");
    }
}

public Item[] list() { return find(null); }
public Item get(String title) throws NotFoundException {
    Item items[] = find(title);
    if (items.length > 0) return items[0];
    throw new NotFoundException();
}
```


B.1 Lösung der Aufgabe 5

- library.server.LibraryDBServant (3/3)

```
private Item[] find(String title) {
    Item[] items = null;
    try {
        int[] itemids = my_db.itemsByTitle(s_title);
        items = new Item[itemids.length];
        for (int i = 0; i < itemids.length; i++) {
            byte[] oid = ItemServantLocator.int2oid(itemids[i]);
            org.omg.CORBA.Object o =
            item_poa.create_reference_with_id(oid,
                ItemHelper.id()); //"IDL:library/Item:1.0";
            items[i] = ItemHelper.narrow(o);
        }
    } catch(Exception ex) {
        ex.printStackTrace();
        return new Item[0];
    }
    return items;
}
```

B.1 Lösung der Aufgabe 5

- library.server.ItemServant

```
package library.server;
import library.*;

public class ItemServant extends ItemPOA
{
    ...
    public synchronized void borrow()
        throws AlreadyBorrowedException {
        if (my_borrowed)
            throw new AlreadyBorrowedException();
        my_borrowed = true;
    }
    public synchronized void giveback()
        throws NotBorrowedException {
        if (!my_borrowed)
            throw new NotBorrowedException();
        my_borrowed = false;
    }
}
```


B.1 Lösung der Aufgabe 5

- library.server.ItemServantLocator (1/4)

```
package library.server;
import ...

public class ItemServantLocator extends ServantLocatorPOA
    // oder: extends LocalObject implements ServantLocator
{
    ORB myorb;
    ItemDatabase mydb;
    Hashtable hash = new Hashtable();

    public ItemServantLocator(ORB orb, ItemDatabase db) {
        myorb = orb; mydb = db;
    }
```


B.1 Lösung der Aufgabe 5

- library.server.ItemServantLocator (2/4)

```
    public static int oid2int(byte[] oid) {
        int val = 0;
        for (int i = 0; i < oid.length; i++)
            val=val*256+oid[i];
        return val;
    }
    public static byte[] int2oid(int val) {
        byte oid[] = new byte[4];
        oid[3] = (byte)(val % 256); val = val / 256;
        oid[2] = (byte)(val % 256); val = val / 256;
        oid[1] = (byte)(val % 256); val = val / 256;
        oid[0] = (byte)(val % 256); val = val / 256;
        return oid;
    }
    // funktioniert im Prinzip auch: (Effizient?)
    // o=new DataOutputStream(bo=new ByteArrayOutputStream())
    // o.writeInt(val); return bo.toByteArray();
    // return (new DataInputStream(new ByteArrayInputStream
    //     (oid))).readInt()
```


B.1 Lösung der Aufgabe 5

- library.server.ItemServantLocator (3/4)

```
public org.omg.PortableServer.Servant preinvoke(
    byte[] oid, POA adapter, String op, CookieHolder cookie)
    throws ForwardRequest
{
    int index oid2int(oid);
    Integer i = new Integer(index);
    ItemServant item;
    synchronized(hash) {
        item = (ItemServant)hash.get(i);
        if (item != null)  // Im Cache vorhanden
            return item;
        Record r = mydb.getItem(index);
        boolean borrowed = false;
        if (r.getData("BORROWED").equals("true"))
            borrowed = true;
        int lendingcount =
            Integer.parseInt(r.getData("LENDINGCOUNT"));
        item = new ItemServant(
            r.getData("TITLE"), borrowed, lendingcount);
        hash.put(i, item);
    }
    return item;
}
```

B.1 Lösung der Aufgabe 5

- library.server.ItemServantLocator (4/4)

```
public void postinvoke(byte[] oid, POA adapter,
    String operation, java.lang.Object the_cookie,
    org.omg.PortableServer.Servant the_servant)
{
    if (the_servant instanceof ItemServant) {
        ItemServant itemservant = (ItemServant)the_servant;
        Record r = mydb.getItem(oid2int(oid);
        r.setData("BORROWED", "false");
        if (itemservant.borrowed())
            r.setData("BORROWED", "true");
        int lcount = itemservant.getLendingcount();
        r.setData("LENDINGCOUNT", Integer.toString(lcount);
    }
    // shrink Cache?
}
```


B.2 Distributed Events

- Probleme
  - Netzwerkausfall
  - Reihenfolge
  - Event-Senke kann verschwinden
- Lösungen:
  - Exceptions
  - Sequenznummer
  - Lease
- zusätzlich: ein Identifikations-Objekt (Handback) von Listener, welches bei jedem Event wieder mitgegeben wird.

1 RemoteEvent

- Basisklasse aller Jini Events

```
package net.jini.core.event;
import java.rmi.MarshalledObject;

public class RemoteEvent
                    implements java.io.Serializable {
    public RemoteEvent( Object source,
                        long eventID,
                        long seqNum,
                        MarshalledObject handback)
    public long getID();
    public long getSequenceNumber();
    public MarshalledObject getRegistrationObject();
}
```

- Enthält: Proxy des Senders, EventID, Sequenznummer, Objekt, welches bei der Registrierung angegeben wurde.

2 RemoteEventListener

- Nur ein Interface, das von einer Event-Senke implementiert werden muß

```
public interface RemoteEventListener extends
            java.rmi.Remote, java.util.EventListener {

    public void notify(RemoteEvent theEvent)
                        throws UnknownEventException,
                           java.rmi.RemoteException;
}
```

- **`UnknownEventException`** bei unbekannten Events (z.B. Quelle und Typ passt nicht zusammen), der Sender sollte daraufhin den Versand dieses Eventtyps einstellen.

3 EventRegistration

- Wird von der Event-Quelle bei der Registrierung zurückgegeben

```
package net.jini.core.event;
import net.jini.core.lease.Lease;
public class EventRegistration
                        implements java.io.Serializable {
    public EventRegistration(  long eventID,
                               Object source,
                               Lease lease,
                               long seqNum);
    public long getID()
    public Object getSource();
    public Lease getLease();
    public long getSequenceNumber();
}
```

- Enthält: EventID, Quelle, Lease, aktuelle Sequenznummer
- **Kein Standard-Interface für Registrierung**!

4 Beispiel

- einfaches Beispiel für einen Listener

```
protected RemoteEventListener listener = null;

public EventRegistration
            addRemoteListener(RemoteEventListener listener)
                throws java.util.TooManyListenersException {
    if (this.listener == null)
        this.listener = listener;
    else
        throw new java.util.TooManyListenersException();
    return new EventRegistration(0L, proxy, null, 0L);
}


protected void fireNotify(long eventID, long seqNum) {
    if (listener == null) return;
   RemoteEvent remoteEvent = new RemoteEvent(proxy, eventID,
                                             seqNum, null);
    listener.notify(remotEvent);
}
```


5 ServiceEvent

- Problem: Wie erfahre ich, wenn sich ein passender Service nach meiner Suche anmeldet?
- Lookup Service verschickt **`ServiceEvents`**
- Anmelden mittels **`notify`**

```
public interface ServiceRegistrar {
    public static final int TRANSITION_MATCH_NOMATCH = ...
    public static final int TRANSITION_NOMATCH_MATCH = ...
    public static final int TRANSITION_MATCH_MATCH = ...

    public EventRegistration
        notify( ServiceTemplate tmpl,
                int transitions,
                RemoteEventListener listener,
                java.rmi.MarshalledObject handback,
                long leaseDuration)
                            throws java.rmi.RemoteException;
    ...
}
```


5 ServiceEvent (2)

- Die Events werden über das Standard-Interface **`net.jini.core.event.RemoteEventListener`** entgegengenommen
- Im Event sind Informationen über die Änderung enthalten
- **`net.jini.core.lookup.ServiceEvent`**

```
public abstract class ServiceEvent
                    extends net.jini.core.event.RemoteEvent
{
    public ServiceID getServiceID();
    public int getTransition();
    public abstract ServiceItem getServiceItem();
}
```

6 Beispiel

- Bei jeder Änderung benachrichtigen lassen:

```
public class RegistrarObserver implements RemoteEventListener {
...
    ServiceRegistrar registrar = ....;
    int transitions = ServiceRegistrar.TRANSITION_MATCH_NOMATCH |
                      ServiceRegistrar.TRANSITION_NOMATCH_MATCH |
                      ServiceRegistrar.TRANSITION_MATCH_MATCH;

    RemoteEventListener proxy =
                (RemoteEventListener) exporter.export(this);

    ServiceTemplate template = new ServiceTemplate(null,null,null);

    try {
     registrar.notify(template, transitions,
                      proxy, null, Lease.ANY);
    } catch(RemoteException e) { e.printStackTrace();}
...
}
```


6 Beispiel (2)

- ServiceEvent auswerten:

```
public void notify(RemoteEvent evt)
                throws RemoteException, UnknownEventException {
    try {
        ServiceEvent sevt = (ServiceEvent) evt;
        switch (sevt.getTransition()) {
           case ServiceRegistrar.TRANSITION_NOMATCH_MATCH:
               System.out.println("nomatch -> match");break;
           case ServiceRegistrar.TRANSITION_MATCH_MATCH:
               System.out.println("match -> match"); break;
           case ServiceRegistrar.TRANSITION_MATCH_NOMATCH:
               System.out.println("match -> nomatch");break;
        }
        if (sevt.getServiceItem() == null) {
           System.out.println("now null");
        } else {
           Object service = sevt.getServiceItem().service;
           System.out.println("Service is " + service.toString());
        }
    } catch(Exception e) {e.printStackTrace(); }       }
```


7 Jini Browser

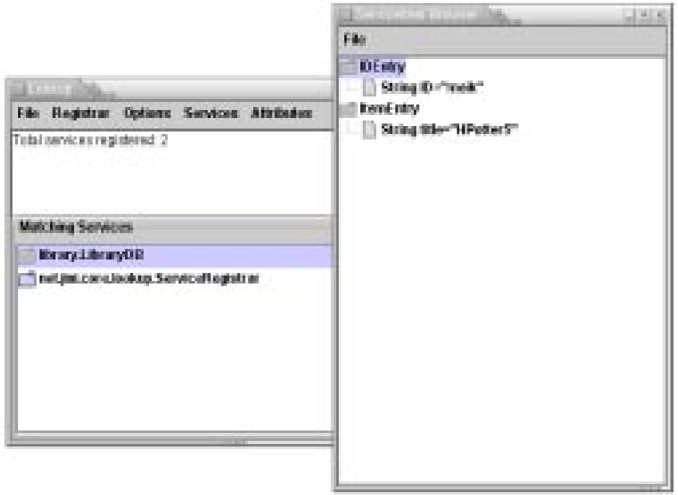
- Programm, um in Lookup Services zu suchen und aufzulisten
- Browser starten

```
java -cp ${JINI_HOME}/lib/browser.jar
    -Djava.security.policy=my.policy
    -Djava.rmi.server.codebase=
                        http://webserver:4711/browser-dl.jar
                        com.sun.jini.example.browser.Browser
```

  - Browser registriert sich für ServiceEvents
  - Lookup Service braucht Proxy, um Browser Events zuzustellen
  - Proxy-Code muss über WWW-Server geladen werden
- Web-Server starten

```
java -jar ${JINI_HOME}/lib/tools.jar
        -port 4711 -dir ${JINI_HOME}/lib/ -verbose
```


7 Jini Browser (2)



B.3 Security-Policies

- Legen das Verhalten des SecurityManagers fest
- Standard Policy in: **`$JAVA_HOME/jre/lib/security/java.policy`**
- Benutzer Policy in: **`$HOME/.java.policy`**
- Zusätzliche Policy wird mit der Eigenschaft **`java.security.policy`** angegeben

```
java -Djava.security.policy=URL Klassenname
```


1 Policy-File Syntax

- Schlüsselquelle angeben (erzeugen mit Hilfe von **`keytool`**)

```
keystore "keystore_url", "keystore_type";
```

- Zugriff erlauben

```
grant signedBy "signer_name1,s_n2,..", codeBase "URL" {
   permission permission_class_name "target_name", "action",
         signedBy "signer_names";
    ....
};
```

- Wem wird ein Recht gegeben
  - ◆ signedBy: das Recht nur signiertem Code (aus signiertem JAR-File) geben
  - ◆ codeBase: nur Code von einer bestimmten Quelle bekommt das Recht
- Welches Recht wird vergeben
  - ◆ spezifiziert durch Permission-Klasse und Argumente
  - ◆ signedBy: die Permission Klasse selbst muss signiert sein

2 Beispiele

- Jedem alles erlauben:

```
grant {
      permission java.security.AllPermission "", "";
}
```

- Erlaubnis abhängig von der Herkunft des Codes

```
grant codebase "file:/local/java-lib/jini-2.0/lib/*" {
    permission java.security.AllPermission;
};
```

- ◆ Endung bestimmt, welchen Klassen die Erlaubnis gewährt wird:
  - **`.../lib/`** = alle Klassen, keine JAR-Archive
  - **`.../lib/*`** = alle Klassen und JAR-Archive
  - **`.../lib/-`** = alle Klassen und JAR-Archive, auch in Unterverzeichnissen

2 Beispiele (2)

- Nur bestimmte Rechte geben

```
grant {
   permission java.net.SocketPermission
                  "129.132.200.35", "connect,accept";
}
```

- genau spezifizierte Rechte

```
grant signedBy "sysadmin", codeBase "file:/home/sysadmin/*"{
   permission java.security.SecurityPermission
                         "Security.insertProvider.*";
    permission java.security.SecurityPermission
                         "Security.removeProvider.*";
    permission java.security.SecurityPermission
                         "Security.setProperty.*";
  };
```

B.4 RMI Proxies

1 Traditionelles RMI

- implizit:

```
public class x extends UnicastRemoteObject implements Remote
{ ... }
```

- explizit:

```
Epublic class x implements Remote {
    public static void main(String[] args) throws Exception {
        Remote demo = new x();
        // this exports the RMI stub to the Java runtime
        RemoteStub stub =
                    UnicastRemoteObject.exportObject(demo);
    }
}
```


2 Exporter

- Nachteil des traditionellen RMI:
  - kein explizites "unexport"
  - keine Auswahl des zu verwendenden Protokolls: RMI over HTTP, IIOP, SSL
  - expilizites Erzeugen des Stubs und Skeletons
- Exporter
  - explizites Exportieren und "Unexportieren"
  - verschiedene Aufrufprotokolle
  - Beispiel:

```
Exporter exporter1 = new JrmpExporter();
Exporter exporter2 = new IiopExporter();

Remote proxy = exporter1.export(demo);

exporter1.unexport(true);
```

  - Ein Exporter ist imer nur für ein Objekt verantwortlich

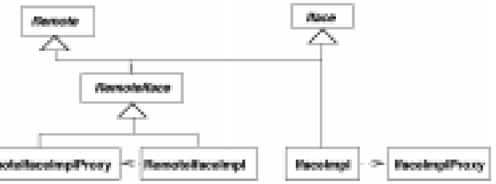
3 Jeri Exporter

- **J**ini **E**xtensible **R**emote **I**nvocation
  - Jini Trust-Modell
  - Stubs müssen nicht mehr explizit erzeugt werden
  - verschiedene Transportprotokolle (HTTP, HTTPS, SSL, TCP)
  - flexibler GC
  - bessere Konfigurierbarkeit
- Konfiguration mittels zweier Parameter
  - ServerEndpoint: Kommunikationsendpunkt des gewünschten Protokolls
  - InvocationLayerFactory: Stub-, Skeleton Generator
- Beispiel:

```
Exporter exporter = new BasicJeriExporter(
    TcpServerEndpoint.getInstance(0), new BasicILFactory());
```


3 Jeri: Exportierte Interfaces

- Der Proxy implementiert alle Remote-Interfaces des Originalobjekts
- Beispiel:



  - **IfaceImplProxy** implementiert keine Innterfaces!

```
Iface iface = (Iface) ifaceImplProxy      // class cast error
Iface iface = (Iface) ifaceRemoteImplProxy  // okay
```

4 Konfiguration

- Konfiguration
  - des Transport-Protokolls (z.B. TCP, Firewire)
  - des Aufrufprotokolls (x.B. IIOP, JRMP, Jeri)

  zur Laufzeit möglich

- Schnittstelle zu Konfigurationsdaten:

```
public interface Configuration {
    Object getEntry(  String component,
                      String name, Class type)
...
}
```

  - Ein Konfigurationseintrag besteht aus: Komponenten Name, Name und Klasse des zu konfugurierenden Objekts
  - Implementiert z.B. in **`net.jini.config.ConfigurationFile`**

4 Konfiguration (2)

- **`Configuration`** Objekt von einem **`ConfigurationProvider`** erzeugt
  - ohne weitere Angaben wird ein **`ConfigurationFile`** Objekt erzeugt
  - Beispiel:

```
Configuration config =
    ConfigurationProvider.getInstance(
                            new String[]{"myconfig.file"});
```

- In einer Konfigurationsdatei kann man dann die gewünschte Klasse angeben

- Format einer Konfigurationsdatei:

```
import net.jini.jrmp.*;
KomponentenName {
    Objektname = new Klassenname();
}
```


4 Konfiguration (3)

- Beispiel einer Konfigurationsdatei:

```
import net.jini.jrmp.*;
import net.jini.iiop.*;
import net.jini.jeri.BasicILFactory;
import net.jini.jeri.BasicJeriExporter;
import net.jini.jeri.tcp.TcpServerEndpoint;

JrmpExportDemo {
    exporter = new JrmpExporter();
}
IiopExportDemo {
    exporter = new IiopExporter();
}
JeriExportDemo {
    exporter = new BasicJeriExporter(
                        TcpServerEndpoint.getInstance(0),
                        new BasicILFactory());
}
```


4 Konfiguration - Beispiel

- Konfiguration aus Datei laden

```
import java.rmi.*;
import net.jini.config.*;
import net.jini.export.*;

String CONFIG_FILE = "jeri/jeri.config";
String[] configArgs = new String[] {CONFIG_FILE};

// get the configuration
Configuration config =
            ConfigurationProvider.getInstance(configArgs);

// use this to construct an exporter
Exporter exporter = (Exporter) config.getEntry(
                                        "JeriExportDemo",
                                        "exporter",
                                        Exporter.class);
// export an object of this class
Remote proxy = exporter.export(new ConfigExportDemo());
```

B.5 JINI im CIP Pool

- JINI Starter Kit unter **`/local/java-lib/jini-2.0`**
- Environment (tcsh)

```
setenv JINI_HOME /local/java-lib/jini-2.0
setenv CLASSPATH .:$JINI_HOME/lib/jini-core.jar
                              :$JINI_HOME/lib/jini-ext.jar
```

- Reggie

```
/proj/i4mw/pub/aufgabe6/reggie/reggie.sh
```

◆ um Reggie zu nutzen benötigt man eine Codequelle (z.B. WebServer)

- WebServer:

```
java -jar $JINI_HOME/lib/tools.jar -port 8081 -dir . -verbose
```

- eigene Anwendung starten

```
java    -Djava.security.policy=mystart.policy
        -Djava.rmi.server.codebase=http://mywebserver:port/
        <Klassenname>
```