



## ***Netzprogrammierung CORBA***

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

# Überblick

1. OMG/CORBA
2. IDL
3. Java IDL Mapping

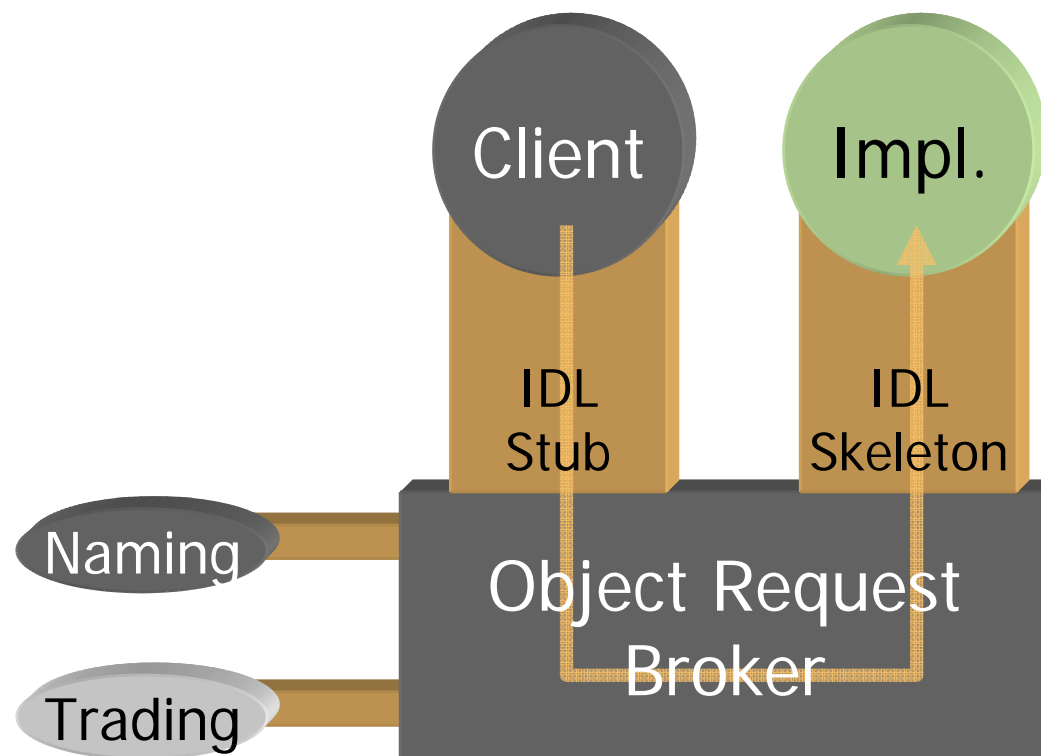
- RMI ist sprachgebundene Plattform von Sun
- *CORBA* (Common Object Request Broker Architecture):
  - offene
  - sprachunabhängige
  - herstellerunabhängige

Plattform für vernetzte Anwendungen

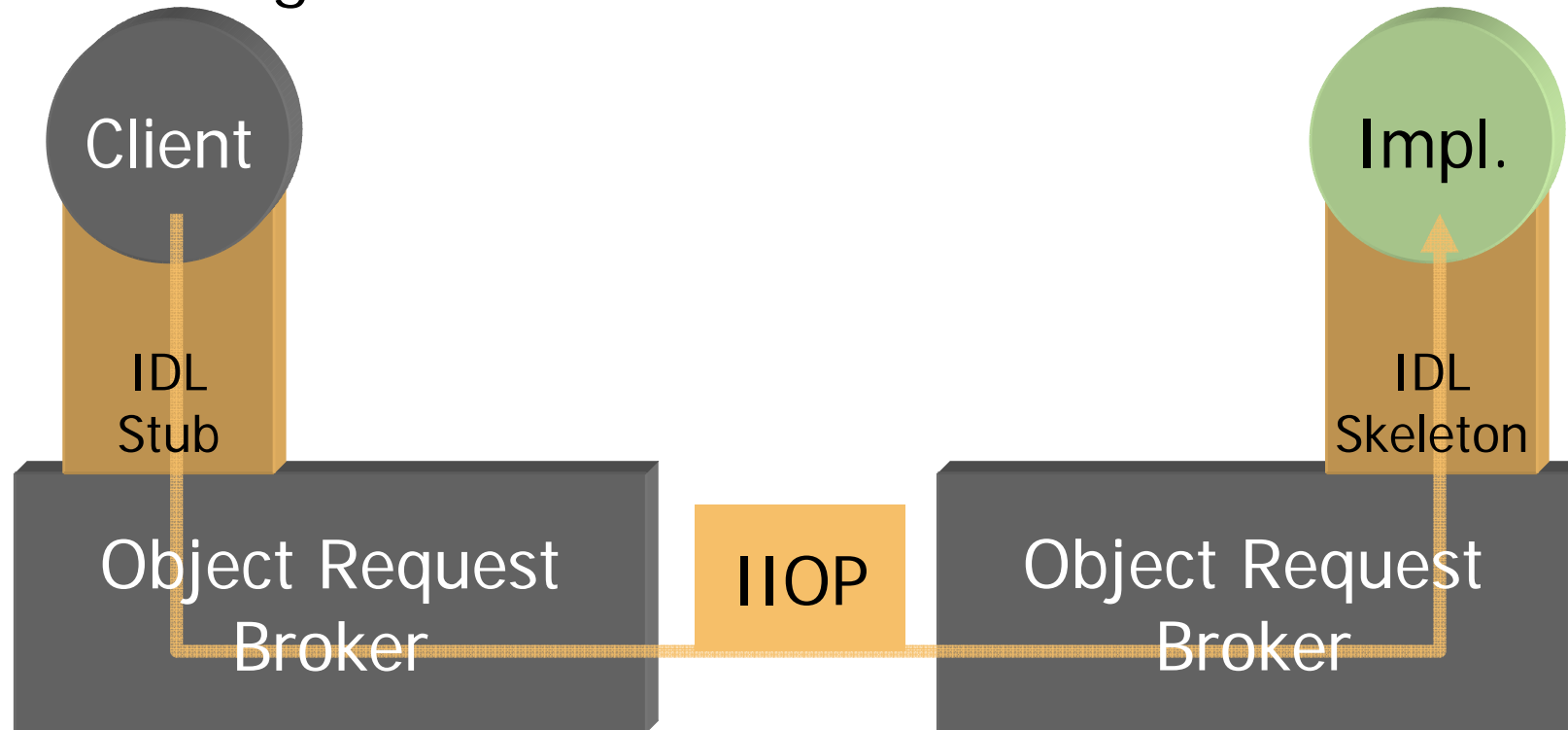
- Träger: Object Management Group *OMG*
  - „The Object Management Group (OMG) is an open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications.“

- Beispiele von OMG Standards
  - UML – Unified Modelling language
  - CORBA – Common Object Request Broker Architecture
  - CWM – Common Warehouse Metamodel
  - MDA – Model Driven Architecture
- Zusammenarbeit mit ISO
- Mitglieder
  - *alle* relevanten großen und kleinen IT Hersteller
- [www.omg.org](http://www.omg.org)
- OMG-CORBA ist der Industriestandard für die Infrastruktur plattformübergreifender verteilter Anwendungen

- CORBA Objekte sind Bausteine von Anwendungen
- Sie haben eine typisierte Schnittstelle
- Von der Schnittstelle getrennte Implementierungen in unterschiedlichen Programmiersprachen
- Aufrufe werden durch Object Request Broker transportiert
- Weitere Dienste unterstützen



- Ortstransparenz wird durch Internet Inter-ORB Protocol IIOP erzeugt



- Interworking zwischen ORBs unterschiedlicher Hersteller möglich

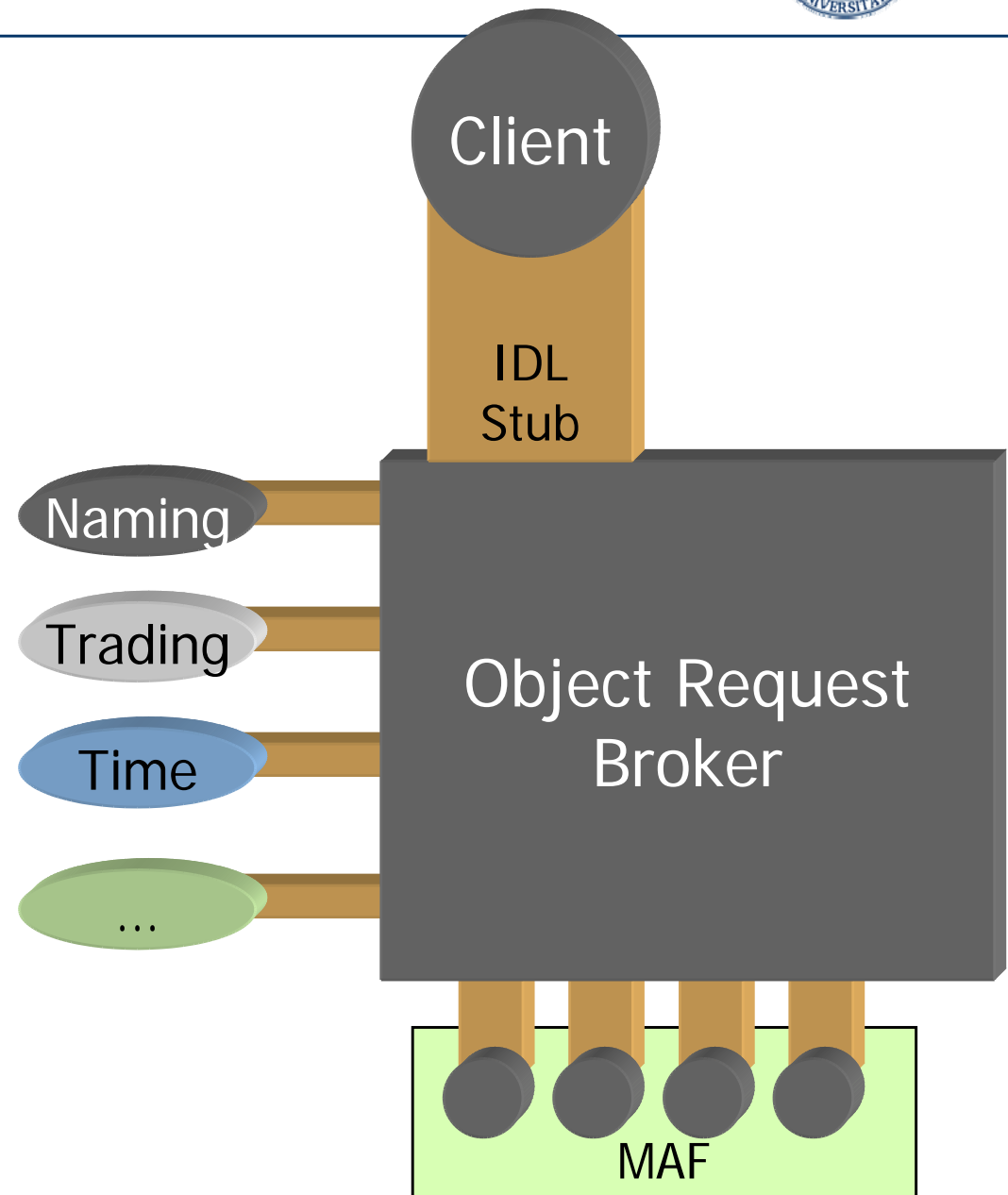
# CORBA Bestandteile

---

- Typisierung von Schnittstellen:
  - Interface Definition Language IDL
  - Vergleiche: Java-Teilsprache für Schnittstellen
  - Abbildungen von IDL zu Programmiersprachen
- APIs zum Objektaufruf (statisch und dynamisch)
  - Vergleiche: Java-RMI Pakete
- Implementierung eines ORB als Transportschicht
  - Vergleiche: Java RMI Laufzeitsystem
- Spezielle Objekte
  - CORBA Services: Verteilt angebotene niedere Dienste (z.B. Collection Service, Time Service, Externalization Service, Concurrency Service etc.)
  - Vergleiche: Java Klassenbibliotheken
  - Common Facilities: Nützliche anwendungsbezogene Dienste, die aber nicht als notwendige Dienste angeboten werden müssen. Heutiger Stand: Internationalization, Print Facility, Mobile Agent Facility
  - Vergleiche: javax Pakete

# CORBA Produkte

- CORBA Bestandteile sind einzeln erhältlich
  - ORBs
  - IDL Anbindungen
  - Services
  - MAF
- Durch CORBA können diese Komponenten unterschiedlicher Herkunft und Implementierung zusammenarbeiten



# CORBA Objekte

---

- Eine mit IDL beschriebene Schnittstelle  
... kann *verschiedene Implementierungen* ...  
... in *verschiedenen Programmiersprachen* haben...  
... aus denen sich *verschiedene Exemplare* ergeben...  
... die als *CORBA Objektreferenz* zugänglich sind ...  
... und *über ORB aufgerufen* werden können.

- Object Management Group *OMG* entwickelt Standards
- Object Management Architecture *OMA* als Standard für verteiltes offenes Objektsystem
- Object Request Broker *ORB* als Laufzeitsystem darin
- Common Object Request Broker *CORBA* als abstrakte Spezifikation von ORBs
- Interface Definition Language *IDL* als Sprache zur Beschreibung von Schnittstellen in OMA
- Internet Inter-ORB Protocol *IIOP* als Protokoll zwischen ORBs
- *ORB* ist die eigentliche Middleware
- *Stub* ist Stellvertreter für entferntes Objekt
- *Skeleton / Adapter* ist Verwaltung des aufrufbaren Objekts
- *Portable Object Adapter POA* als Adapter Muster
- *IDL* beschreibt Schnittstellen
- *IDL Compiler* erzeugt Stub und Skeleton und jeweiliges Interaktion mit ORB in Programmiersprache



## Beispiel

# Counter Beispiel

---

- Zähler Objekt als CORBA Service anbieten und nutzen
- Kann Zähler erhöhen, kann Zählerstand zu einem Parameter addieren und Auskunft über die Anzahl der Änderungen geben
- Notwendig:
  - Schnittstellendefinition
  - Zählerobjekt
  - Registrierung des Servers
  - Nutzung des Servers durch Client
- CORBA ist Teil des Java SDK Standard Edition
- Dort aber nur in Teilen implementiert

# Schnittstellendefinition in IDL

```
module counter {
```

```
    typedef struct Info {  
        long value;  
        long counted;  
    } _Info;
```

Eigener  
Typ

```
    interface Counter {  
        readonly attribute long value;  
        void add(in long value);  
        void addTo(inout long clientValue);  
        Info getInfo();  
    };  
};
```

Schnittstelle

Signatur

Namensraum

# Erzeugung von Stubs etc

---

- `>idlj -fall counter.jdl`
- Erzeugt in einem Paket `counter` verschiedene Klassen
- Abbildung Info Typ auf Java-Klasse
  - `Info.java`: Java Klasse
  - `InfoHelper.java`: Marshalling der Info Objekte
  - `InfoHolder.java`: Hilfsklasse für Info als inout Parameter
- Stubs und Skeleton für Counter
  - `CounterOperations.java`: Counter Interface
  - `Counter.java`: Typisierung als CORBA Interface
  - `CounterHelper.java`: Marshalling der Counter Objekte
  - `CounterPOA.java`: Server Skeleton (Object Adapter)
  - `_CounterStub.java`: Stub für Client
- `>javac counter/* .java`

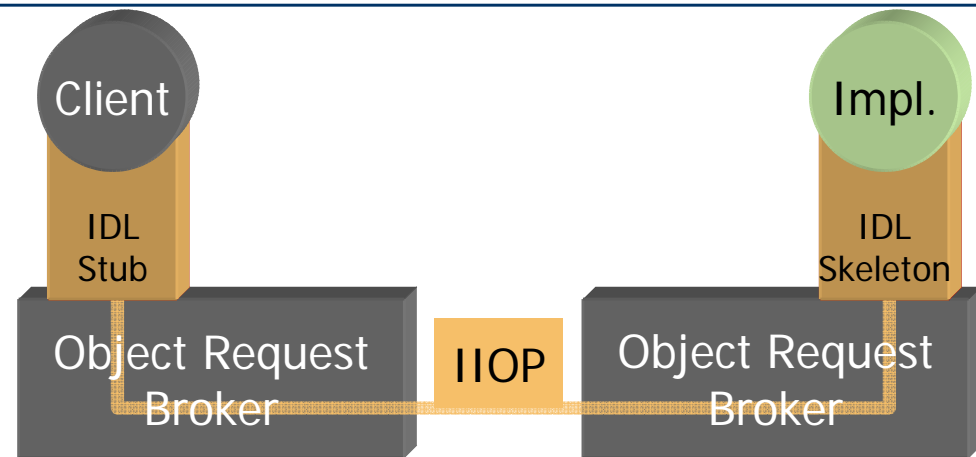
# Counter Objekt

```
package counter;
public class CounterServiceImpl extends CounterPOA {
    int value = 0;                // lokale Felder
    int addedCounter = 0;

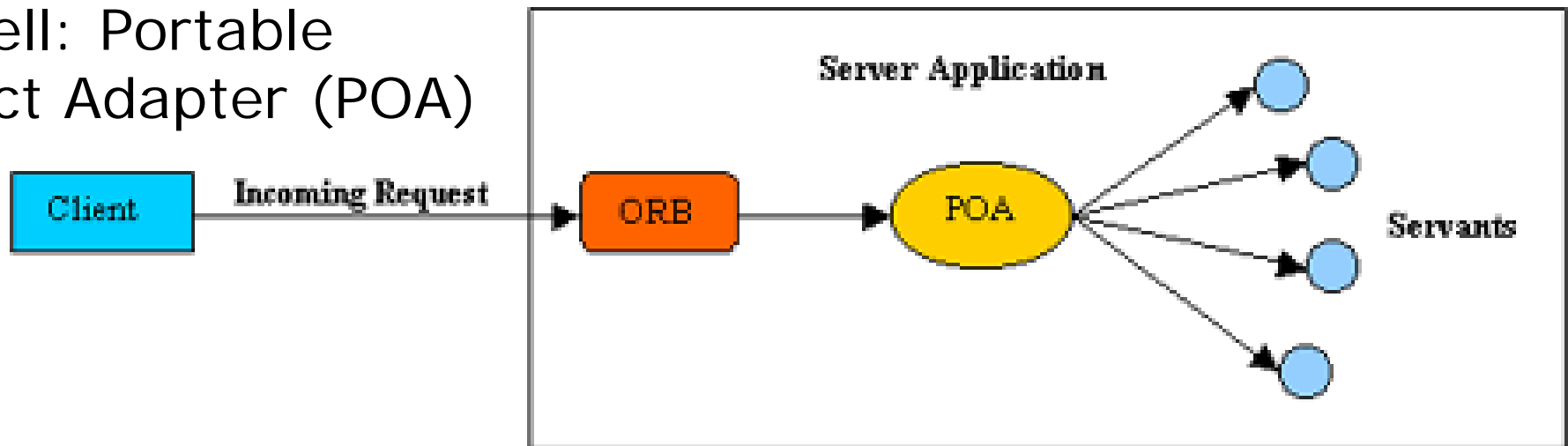
    public int value() {return value;}           // value Attribut herausgeben
    public void add (int value) {               // Zähler erhöhen
        this.value += value;
        addedCounter ++;
    }
    // Zähler zu Parameter addieren
    public void addTo(org.omg.CORBA.IntHolder clientValue) {
        clientValue.value += value;
    }
    public Info getInfo() {                   // Info abliefern
        Info info = new Info();
        info.value = value;
        info.counted = addedCounter;
        return (info);
    }
}
```

# CounterPOA ist ein Object Adapter

- Konzeptionell:
- Zur Anpassung von Objekten sieht CORBA Objektadapter als Schnittstelle zwischen Programm und ORB vor
  - ORB kann Objekte in unterschiedliche Sprachen integrieren
  - Also Adapter notwendig



- Aktuell: Portable Object Adapter (POA)



- 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();
```

Narrow wäre in Java ein Cast:  
(POA) orb.resolve...

- ORB ist aus `org.omg.CORBA.ORB`
- POA ist aus dem Paket `org.omg.PortableServer`

# Counter Server Programm (1)

---

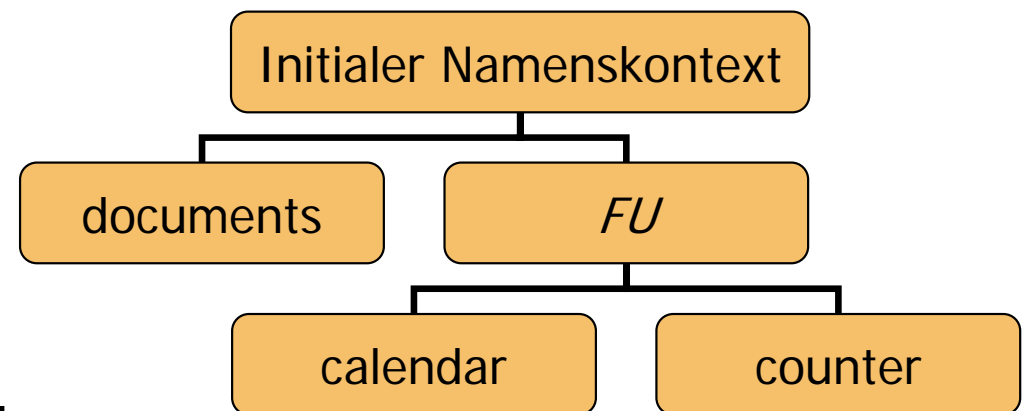
```
package counter;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.PortableServer.*;

public class CounterServer {
    public static void main(String[] args) throws Exception {
        // ORB initialisieren
        ORB orb = ORB.init(args,null);
        // Root Objekt ermitteln und aktivieren
        org.omg.CORBA.Object rootRef =
            orb.resolve_initial_references("RootPOA");
        POA root = POAHelper.narrow(rootRef); // Cast (POA) rootRef
        root.the_POAManager().activate();
    }
}
```

- Der Namensdienst ist ein Standarddienst in CORBA
- Er ist definiert durch eine IDL und als OMG Standard festgelegt
- Seit 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](#)

# 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
- RMI-URLs sind Zeichenketten
- CORBA Namenspfade sind Objekte



# Binden eines Namens

---

- Initialen Namenskontext abfragen

```
NamingContextExt context =  
    NamingContextExtHelper.narrow(  
        orb.resolve_initial_references("NameService"));
```

- Namensobjekt erzeugen

```
NameComponent name1[] =  
    context.to_name("documents");
```

- Dummy-Objekt an den Namen binden

```
context.rebind(name1, someObjectARef);
```

# Neuen Kontext einfügen

- Neuen Kontext erzeugen und binden

```
NameComponent name2[] = context.to_name("FU");  
NamingContextExt context2 =  
    (NamingContextExt)context.bind_new_context(name2);
```

- Weitere Bindungen einfügen

```
NameComponent name3[] = context.to_name("calendar");  
context2.rebind(name3, someObjectBRef);  
NameComponent name4[] = context.to_name("counter");  
context2.rebind(name4, counterRef);
```

# Bindung auflösen

---

- NameService erhalten und Namen auflösen

```
NamingContextExt nc =  
    NamingContextExtHelper.narrow(  
        orb.resolve_initial_references("NameService"));  
org.omg.CORBA.Object docs = nc.resolve_str("documents");  
org.omg.CORBA.Object calendar =  
    nc.resolve_str("FU/calendar");  
org.omg.CORBA.Object count =  
    nc.resolve_str("FU/counter");
```

## Counter Server Programm (2)

```
// Namensdienst ermitteln
org.omg.CORBA.Object namingRef =
    orb.resolve_initial_references("NameService");
NamingContextExt naming =
    NamingContextExtHelper.narrow(namingRef); // (NamingContextExt)
// Neuen Counter erzeugen
CounterServiceImpl theCounter = new CounterServiceImpl();
// Referenz auf Counter Objekt unter "Counter" registrieren
org.omg.CORBA.Object serviceRef =
    root.servant_to_reference(theCounter);
NameComponent[] path = naming.to_name("Counter");
naming.rebind(path,serviceRef);
// Auf Aufrufe warten
orb.run();
}
}
```

# Counter Client Programm

---

```
package counter;
import org.omg.CORBA.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;

public class CounterClient {
    public static void main(String[] args) throws Exception {
        // ORB initialisieren
        ORB orb = ORB.init(args,null);
        // Namensdienst auffinden
        org.omg.CORBA.Object namingRef =
            orb.resolve_initial_references("NameService");
        NamingContextExt naming =
            NamingContextExtHelper.narrow(namingRef); //
        (NamingContextExt)
        // Referenz auffinden
        org.omg.CORBA.Object counterRef =
            naming.resolve_str("Counter");
        Counter counter = CounterHelper.narrow(counterRef);
    }
}
```

# Counter Client Programm

```
// Etwas spielen
System.out.println("Wert: " + counter.value());
counter.add(10);
System.out.println("Wert: " + counter.value());
counter.add(10);
System.out.println("Wert: " + counter.value());
IntHolder myInt=new IntHolder(100);
counter.addTo(myInt);
System.out.println("myInt: " + myInt.value);
Info info = counter.getInfo();
System.out.println("Info: " + info.value + " / " + info.counted);
}
}
```

# Verteilt ausführen

```
>orbd -ORBInitialPort 1050 -ORBInitialHost localhost
```

```
>java counter.CounterServer -ORBInitialPort 1050 -ORBInitialHost localhost
```

```
athos >java counter.CounterClient -ORBInitialPort 1050  
-ORBInitialHost 160.45.114.204
```

Wert: 40

Wert: 50

Wert: 60

myInt: 160

Info: 60 / 6

- 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;

```

...

- 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
  - JDK Dokumentation  
Als Javadoc beiliegend

# 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 nameserverhost  
Wo ist der Namensdienst (Normalfall: localhost)
  - -ORBInitialPort nameserverport  
Auf welchem Port arbeitet der Namensdienst (Normalfall: 900)
- Weitere Methoden, beispielsweise zur Termination

# 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);
```

# 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");
```

| <i>RMI</i>  | <i>CORBA</i>  |
|---|---|
| Eine Sprache  | Sprachunabhängig  |
| keine Mappings nötig  | Mappings notwendig  |
| Objekte können als Argumente verwendet werden<br>(Weil Klassen nachladbar sind) | Nur Referenzen auf Objekte möglich                        |
| Ein Laufzeitsystem  | Unterschiedliche Laufzeitsysteme (ORBs) integriert        |
| Proprietär (?)<br>„Eigentümer“: Sun<br>Prozess: JCP                             | Offener Standard (?)<br>„Eigentümer“: OMG<br>Prozess: OMG |

- In Java sind mehrere „Middlewares“ integriert (RMI, CORBA, EJB, ...)



## Interface Definition Language IDL

- IDL ist eine Sprache zur Definition von Schnittstellen
- Sprachunabhängig
- Semantik der Typen definiert
- Zur Nutzung Mapping zu konkreter Sprache notwendig
- OMG definiert solche Mappings für unterschiedlichste Sprachen
- Abbildungsprobleme müssen zur Laufzeit durch Ausnahmen signalisiert werden

# IDL Typen (Ausschnitt)

- Einfache Typen
  - Nicht interpretiertes Byte mit 8 Bit Länge  
octet
  - Ganzzahlen mit 16, 32, 64 Bit Länge, mit/ohne Vorzeichen:  
short                    unsigned short  
long                    unsigned long  
long long                unsigned long long
  - Fließkommazahlen mit einfacher, doppelter und erweiterter Genauigkeit nach IEEE, Festkommazahlen  
float                    double                    long double                    fixed
  - Zeichen für beliebige Zeichen, 8 Bit oder länger  
char                    wchar                    string                    wstring
  - Wahrheitswert (TRUE, FALSE)  
boolean
  - Typplatzhalter  
any

## struct

- Beispiel:

```
struct cumulativeWeatherInfo {  
    long start;    // Beginning of interval in Unix time() format  
    long end;      // End of interval in Unix time() format  
    float sunshine; // Hours of sunshine during interval.  
    float rainfall; // mm of rainfall during interval.  
};
```

- Syntax:

(69)  $\langle \text{struct\_type} \rangle ::= \text{"struct"} \langle \text{identifier} \rangle \text{"\{"}$   
     $\langle \text{member\_list} \rangle \text{"\}"}$

(70)  $\langle \text{member\_list} \rangle ::= \langle \text{member} \rangle +$

(71)  $\langle \text{member} \rangle ::= \langle \text{type\_spec} \rangle \langle \text{declarators} \rangle \text{";"}$

# Zusammengesetzte IDL Typen (Ausschnitt): unions

- Beispiel

```
union U2 switch (char) {  
  case 'a':  
    long a;  
  case 'b':  
  case 'c':  
    short b;  
  default:  
    octet c; };
```

- Syntax:

(72) `<union_type> ::= "union" <identifier> "switch" "(" <switch_type_spec> ")"`  
`"{" <switch_body> "}"`

(73) `<switch_type_spec> ::= <integer_type> | <char_type> |`  
`<boolean_type> |`  
`<enum_type> | <scoped_name>`

(74) `<switch_body> ::= <case>+`

(75) `<case> ::= <case_label>+ <element_spec> ";"`

(76) `<case_label> ::= "case" <const_exp> ":" | "default" ":"`

(77) `<element_spec> ::= <type_spec> <declarator>`

- Beispiel  
`enum color {red, green, blue};`
- Syntax
  - (78) `<enum_type> ::= "enum" <identifier> "{ " <enumerator> { "," <enumerator> }* "}"`
  - (79) `<enumerator> ::= <identifier>`
- Beispiel  
`string StringArray[10];`
- Syntax
  - (83) `<array_declarator> ::= <identifier> <fixed_array_size> +`
  - (84) `<fixed_array_size> ::= "[" <positive_int_const> "]"`
- Beispiel
  - `sequence <float> severalFloats;`
  - `sequence <boolean,10> tenBooleans;`
- Syntax
  - (80) `<sequence_type> ::= "sequence" "<" <simple_type_spec> "," <positive_int_const> ">" | "sequence" "<" <simple_type_spec> ">"`

# Module und Schnittstellen

---

- Modul
  - Namensraum für zusammengehörige Schnittstellen
  - Qualifizierter Name: *Modulname::Name*
  - Vergleichbar Paket in Java
- Schnittstelle
  - Getypter Interaktionspunkt für Objekte
  - Vergleichbar einem Interface in Java
- Schnittstelle enthält
  - Konstante
  - Attribute
  - Typdefinitionen
  - Operationen
  - Ausnahmedefinitionen
  - ...

# Module und Schnittstellen

- Beispiel für Objektschnittstelle mit nur einem Attribute

```
module weather {  
    interface weatherdata {  
        struct cumulativeWeatherInfo {  
            long start;  
            long end;  
            float sunshine;  
            float rainfall;  
        };  
    };  
};
```

- Bezeichner: Zeichen, Ziffern, \_
- *Groß-/Kleinschreibung wird nicht unterschieden*

# Module und Schnittstellen

- Komplettes Modul mit Typdefinition und Schnittstelle mit Operationen und Attributen

```
module counter {  
  
    typedef struct Info {  
        long value;  
        long counted;  
    } _Info;  
  
    interface Counter {  
        readonly attribute long value;  
        void add(in long value);  
        void addTo(inout long clientValue);  
        Info getInfo();  
    };  
};
```

```
<interface_dcl> ::= <interface_header> "{" <interface_body> "}"
<interface_header> ::=
  [ "abstract" | "local" ] "interface" <identifier>
  [ <interface_inheritance_spec> ]
<interface_inheritance_spec>
  ::= ":" <interface_name> { "," <interface_name> }*
<interface_name> ::= <scoped_name>
<interface_body> ::= <export>*
<export> ::=
  <type_dcl> ";"
  | <const_dcl> ";"
  | <except_dcl> ";"
  | <attr_dcl> ";"
  | <op_dcl> ";"
```

# Konstante

- Beispiel

```
const maxValue = 2020;
```

- Syntax

(27) <const\_dcl> ::=

"const" <const\_type> <identifier> "=" <const\_exp>

(28) <const\_type> ::=

<integer\_type>

| <char\_type>

| <wide\_char\_type>

| <boolean\_type>

| <floating\_pt\_type>

| <string\_type>

| <wide\_string\_type>

| <fixed\_pt\_const\_type>

| <scoped\_name>

| <octet\_type>

# Attribute

- Beispiel

```
attribute float radius;
```

```
readonly attribute position_t position;
```

- Syntax

(85)  $\langle \text{attr\_dcl} \rangle ::= \langle \text{readonly\_attr\_spec} \rangle \mid \langle \text{attr\_spec} \rangle$

(104)  $\langle \text{readonly\_attr\_spec} \rangle ::=$

“readonly” “attribute”  $\langle \text{param\_type\_spec} \rangle$

$\langle \text{readonly\_attr\_declarator} \rangle$

(105)  $\langle \text{readonly\_attr\_declarator} \rangle ::=$

$\langle \text{simple\_declarator} \rangle \langle \text{raises\_expr} \rangle$

$\mid \langle \text{simple\_declarator} \rangle \{ \text{“,”} \langle \text{simple\_declarator} \rangle \}^*$

(106)  $\langle \text{attr\_spec} \rangle ::=$

“attribute”  $\langle \text{param\_type\_spec} \rangle \langle \text{attr\_declarator} \rangle$

(107)  $\langle \text{attr\_declarator} \rangle ::=$

$\langle \text{simple\_declarator} \rangle \langle \text{attr\_raises\_expr} \rangle$

$\mid \langle \text{simple\_declarator} \rangle \{ \text{“,”} \langle \text{simple\_declarator} \rangle \}^*$

# Typdefinitionen

---

- Beispiele

```
typedef struct Info {  
    long value;  
    long counted;  
} _Info;
```

```
typedef sequence<Book> BookList;
```

# Operationen

- Beispiel
  - void add(in long value);
  
- Syntax
  - (87)  $\langle \text{op\_dcl} \rangle ::= [ \langle \text{op\_attribute} \rangle ]$   
 $\quad \langle \text{op\_type\_spec} \rangle \langle \text{identifier} \rangle \langle \text{parameter\_dcls} \rangle$   
 $\quad [ \langle \text{raises\_expr} \rangle ] [ \langle \text{context\_expr} \rangle ]$
  - (88)  $\langle \text{op\_attribute} \rangle ::= \text{"oneway"}$
  - (89)  $\langle \text{op\_type\_spec} \rangle ::= \langle \text{param\_type\_spec} \rangle \mid \text{"void"}$

# Parameter- und Operationsattribute

- Parameterattribute

- in  
Parameter wird vom Client zum Server geschickt (by value)
- out  
Parameter wird vom Server zum Client geschickt (Ergebnis)
- inout  
Parameter wird in beide Richtungen geschickt (änderbar)

`void add(in long value);`

`void addTo(inout long clientValue);`

- Operationsattribut

- oneway
  - Andere Semantic: at-most-once
  - Ergebnis void, keine out Parameter, keine Ausnahme
- Normalfall ohne Ausnahme: Exactly once
- Bei Ausnahme: at-most-once

# Ausnahmen

- In CORBA sind zwei Arten Ausnahmen definiert
  - Standardausnahmen:

```
#define ex_body {unsigned long minor; completion_status completed;}
module CORBA {
  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
  ...
}
```
  - Deklarierte anwendungsspezifische Ausnahmen

# Ausnahmen

- Ausnahmedeklaration

- Beispiel

- exception overspending { long amount; };

- Syntax

(86) <except\_dcl> ::=

“exception” <identifier> “{” <member>\* “}”

- Raises Klausel bei Methodendeklaration:

```
interface Wallet {
    exception overspending { long amount; };
    void pay(long prize) raises(overspending);
};
```



## Java IDL Mapping

# IDL Mapping für Java

- Einfach Typen zu geeigneten Primitiven:

|                               |                      |
|-------------------------------|----------------------|
| boolean                       | boolean              |
| char, wchar                   | char                 |
| string, wstring               | String               |
| octet                         | byte                 |
| short, unsigned short         | short                |
| long, unsigned long           | int                  |
| long long, unsigned long long | long                 |
| float                         | float                |
| double                        | double               |
| fixed                         | java.math.BigDecimal |

# IDL Mapping für Java

- IDL

```
void basicints(in short s, in unsigned short us,  
              in long l, in unsigned long ul,  
              in long long ll, in unsigned long long ull);
```

```
void basicfloats(in float f, in double d);
```

```
void basicchars(in char c, in wchar wc,  
               in string s, in wstring ws);
```

```
void basicmisc (in octet o, in boolean b);
```

- Java

```
void basicints (short s, short us, int l, int ul, long ll, long ull);
```

```
void basicfloats (float f, double d);
```

```
void basicchars (char c, char wc, String s, String ws);
```

```
void basicmisc (byte o, boolean b);
```

## sequence -> Array

---

- Listen zu Felder
- IDL

```
typedef sequence <float>          severalFloats;
```

```
typedef sequence <boolean,10> tenBooleans;
```

```
void mapsequence(in severalFloats sf, in tenBooleans tb);
```

- Java

```
void mapsequence (float[] sf, boolean[] tb);
```

# struct -> Klasse

- Verbünde zu Objekte mit Attributen

- IDL

```
typedef struct Info {  
    long value;  
    long counted;  
} Info;
```

- Java

```
public final class Info implements  
    org.omg.CORBA.portable.IDLEntity {  
    public int value = (int)0;  
    public int counted = (int)0;  
    public Info () { } // ctor  
    public Info (int _value, int _counted)  
    {  
        value = _value;  
        counted = _counted;  
    } // ctor  
} // class Info
```

## enums -> Klasse

---

- Aufzählungen zu Aufzählungsobjekte
- IDL

```
typedef enum color {red, green, blue} _color;  
void mapenum(in color c);
```

# enums -> Klasse

- Java

```
void mapenum (types.tyepetestsPackage.color c);
```

```
private int __value;
```

```
private static int __size = 3;
```

```
private static types.tyepetestsPackage.color[] __array =  
    new types.tyepetestsPackage.color [__size];
```

```
public static final int _red = 0;
```

```
public static final types.tyepetestsPackage.color red=new  
    types.tyepetestsPackage.color(_red);
```

```
public static final int _green = 1;
```

```
...
```

```
public int value () { return __value; }
```

```
public static types.tyepetestsPackage.color from_int (int value)
```

```
{ if (value >= 0 && value < __size) return __array[value];
```

```
    else  
        ();  
    throw new org.omg.CORBA.BAD_PARAM
```

```
}
```

```
protected color (int value)
```

```
{ __value = value;
```

```
    __array[__value] = this;
```

```
}
```

# union -> Klasse

---

- Unions nach Unionobjekte
- IDL

```
typedef union borc switch (short) {  
case 0: short b;  
case 1: long c;  
} _borc;  
void mapunion(in borc b);
```

# union -> Klasse

- Java

```
void mapunion (types.typedestsPackage.borc b);
```

```
public final class borc implements
    org.omg.CORBA.portable.IDLEntity
{ private short ___b;
  private int ___c;
  private short __discriminator;
  private boolean __uninitialized = true;
```

```
public borc () { }
public short discriminator () {... }
public short b () {...}
public void b (short value) {... }
public int c () {... }
public void c (int value). {... }
```

# Operationen

---

- Operationen nach Methoden
- Exceptions nach Exceptions
- Parameterübergabe
  - in Parameter: Normale Werteparameter in Java
  - out und inout Parameter: keine Entsprechung in Java
  - Werden durch Holder-Klassen realisiert
  - Parameter sind dann Objekte dieser Klassen

# Holder Klassen

---

- Vordefiniert für vordefinierte Typen
- org.omg.CORBA.FloatHolder:
  - Field Summary
    - float value  
The float value held by this FloatHolder object.
  - Constructor Summary
    - FloatHolder()  
Constructs a new FloatHolder object with its value field initialized to 0.0.
    - FloatHolder(float initial)  
Constructs a new FloatHolder object for the given float.
- Entsprechend int, long etc.etc.

# Aus Counter Beispiel

- Aus IDL:  
void addTo(inout long clientValue);
- Beim Serverobjekt:  
// Zähler zu Parameter addieren  
public void addTo(org.omg.CORBA.IntHolder clientValue) {  
 clientValue.value += value;  
}
- Beim Klientenobjekt Parameter erzeugen:  
IntHolder myInt = new IntHolder(100);  
counter.addTo(myInt);  
// veränderter Wert in myInt.value

# Holder Klassen

- Automatisch erzeugt bei eigenen Typen

```
package counter;
/**
 * counter/InfoHolder.java .
 * Generated by the IDL-to-Java compiler (portable), version "3.1"
 * from counter.idl
 * Mittwoch, 22. Oktober 2003 15.15 Uhr CEST
 */
public final class InfoHolder implements
    org.omg.CORBA.portable.Streamable
{
    public counter.Info value = null;

    public InfoHolder () { }

    public InfoHolder (counter.Info initialValue)
    { value = initialValue; }

    [...]
}
```

# Literatur

---

- ISO/IEC 14750:1999 Information technology - Open Distributed Processing - Interface Definition Language. JTC 1/SC 7. 1999.
- OMG. CORBA 3.0 Spezifikation.  
[http://www.omg.org/technology/documents/formal/corba\\_2.htm](http://www.omg.org/technology/documents/formal/corba_2.htm)  
1150 Seiten!
- Open Source CORBA Implementierungen:  
[http://sourceforge.net/softwaremap/trove\\_list.php?form\\_cat=51](http://sourceforge.net/softwaremap/trove_list.php?form_cat=51)
- Dokumentation des JSDK 1.4SE



## Zusammenfassung

## 1. OMG/CORBA

1. Sprachunabhängiges Objektmodell
2. ORB als Transportschicht
3. IDL für Schnittstellen
4. Services, Facilities als definierte Dienstobjekte
5. ORB
6. POA
7. NameService

## 2. IDL

1. Einfache Typen
2. Zusammengesetzte Typen
3. Module und Schnittstellen
4. Operationen

## 3. Java IDL Mapping

1. IDL Konzepte teilweise direkt abgebildet
2. IDL Konzepte teilweise auf Helferklassen abgebildet