

BP 1	Fernauf: Literatur, Untergliederung
3	Fernauf
<input type="checkbox"/>	Literatur <i>Nelson, B. J.: Remote Procedure Call. Tech. Rep. CSL-81-9, Xerox Palo Alto Research Center, Palo Alto, Calif., 1981.</i> <i>Kaiserswerth, M.: Der Fernauf als Betriebssystemdienst. Arbeitsberichte des IMMD, Band 22, Nummer 1, Januar 1989.</i> <i>SUN: Network Programming Guide. 1990.</i>
<input type="checkbox"/>	Untergliederung
3.1	Der Begriff, Anwendungsbeispiele
3.2	Überblick über die wesentlichsten Gesichtspunkte
3.3	Ideale Eigenschaften eines transparenten Mechanismus und ihre Realisierung
3.4	Beispiel 1: SUN RPC
3.5	Beispiel 2: Java RMI
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.1-1

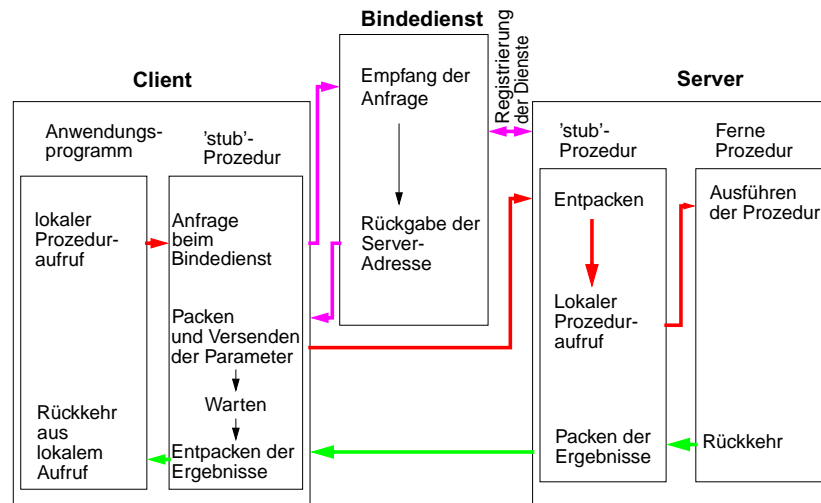
BP 1	Fernauf: Der Begriff, Anwendungsbeispiele
3.1	Der Begriff, Anwendungsbeispiele
<input type="checkbox"/>	Fernauf ist der synchrone Transfer von Kontrolle und Daten zwischen Teilen eines über disjunkte Adreßräume verteilten Programms.
<input type="checkbox"/>	Um in einer prozeduralen Programmiersprache den gleichförmigen Zugriff auf lokale und nicht-lokale Ressourcen zu ermöglichen, sollte sich der Fernauf auf der semantischen (und syntaktischen) Ebene genauso wie ein lokaler Prozedurauf verhalten.
<input type="checkbox"/>	Koroutinen, Ausnahmen (exceptions): <ul style="list-style-type: none"> • Sind ebenfalls Arten eines synchronen Kontroll- und Datentransfers
<input type="checkbox"/>	Synchroner Transfer und Nebenläufigkeit: <ul style="list-style-type: none"> • Unabhängige Konzepte
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.1-2

BP 1	Fernauf: Der Begriff, Anwendungsbeispiele
<input type="checkbox"/>	Anwendungsbeispiele <ul style="list-style-type: none"> • Nelson (Fig. 2.1): Netzwerk unsichtbar für Client und Server; bei der Programmierung Netzwerk ohne Bedeutung; übliche Programmentwicklungsumgebung benutzbar; Test kann auf Monoprozessor erfolgen • File Server: Für Client irrelevant, welche Teile remote sind
<input type="checkbox"/>	Ideale Vorstellung
	<div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="text-align: center;"> Maschine C <div style="border: 1px solid black; padding: 5px; margin: 5px;"> MODULE M_C PROCEDURE P_C[...] ... z ← P_S[x, y] </div> </div> <div style="text-align: center;"> Maschine S <div style="border: 1px solid black; padding: 5px; margin: 5px;"> MODULE M_S PROCEDURE P_S[r, s] ... RETURN[t] </div> </div> </div> <div style="text-align: center; margin-top: 10px;"> </div>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.1-3

BP 1	Fernauf: Der Begriff, Anwendungsbeispiele
<input type="checkbox"/>	Grundmechanismus <ul style="list-style-type: none"> • Stubs; Aufrufender und Aufgerufener bedürfen keiner Änderung
	<div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="text-align: center;"> Client <div style="border: 1px solid black; padding: 5px; margin: 5px;"> MODULE Reader buffer ← ReadPage[file, page] </div> <div style="border: 1px solid black; padding: 5px; margin: 5px;"> ... STUB ... MODULE FileOps ReadPage: STUB[File, Page] RETURNS[Buffer] </div> </div> <div style="text-align: center;"> File Server <div style="border: 1px solid black; padding: 5px; margin: 5px;"> MODULE FileOps ReadPage: PROC[File, Page] RETURNS[Buffer] </div> <div style="border: 1px solid black; padding: 5px; margin: 5px;"> ... STUB ... MODULE FileOpsServer ReadPageServer: STUB[...] </div> </div> </div> <div style="text-align: center; margin-top: 10px;"> </div>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.1-4

BP 1 Fernaufruf: Der Begriff, Anwendungsbeispiele

• Gesamtstruktur



BP 1 Fernaufruf: 2.3 Fernaufruf

- Modellierung des physikalischen Kommunikationssystems einschließlich seines tolerierbaren Fehlverhaltens, Spezifikation Send-Receive
- ◆ Modellierung des Kommunikationssystems, Zustand M

• Nachrichtenverspätung M Menge

• Nachrichtenverlust

$\{\}$
 $send(m)$
 $\{M = 'M \vee M = 'M \cup \{m\}\}$

BP 1 Fernaufruf: 2.3 Fernaufruf

• Nachrichtenverdopplung, Nachrichtenverfälschung

$\{\}$
 $[status, m] \leftarrow receive()$
 $\{ \text{status} = \text{good} \wedge ((M = 'M \vee M = 'M - [m]) \wedge m \in 'M$
 $\})$
 $\vee \text{status} = \text{bad} \wedge (M = 'M \vee \exists m_{\text{bad}} \in 'M (M = 'M - \{m_{\text{bad}}\}))$
 $\}$

◆ Zu berücksichtigende Ausfälle

- Einzelrechner: Prozessorausfall
Prozeßzusammenbruch
- Verteiltes System: Zusätzlich Verbindungsausfall

BP 1 Fernaufruf: Überblick über die wesentlichsten Gesichtspunkte

3.2 Überblick über die wesentlichsten Gesichtspunkte

□ Aufrufsemantik

- Genau-einmal (exactly-once)
- Letzte-Ausführung (last-one)

□ Nebenläufigkeit, Ausnahmebehandlung

- Ausnahmen wie üblich behandeln
- Ergänzung um netzwerkspezifische Ausnahmereisachen

□ Gesichtspunkte der Transparenz auf Sprachebene

- Binden und Konfigurieren
Gesonderte Konfigurierungsbeschreibung
Schnittstellenbeschreibung
- Typprüfung
Heterogene Systeme
- Parameterfunktionalität
Marshaling

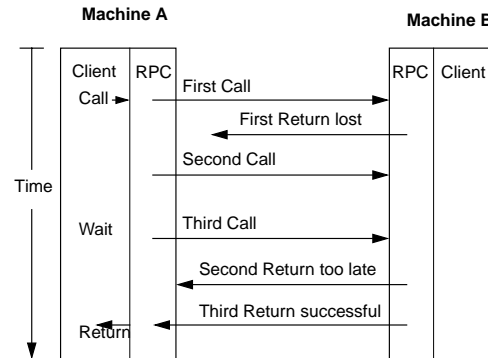
BP 1 Fernaufruf: Ideale Eigenschaften eines transparenten Mechanismus

3.3 Ideale Eigenschaften eines transparenten Mechanismus

Aufrufsemantik

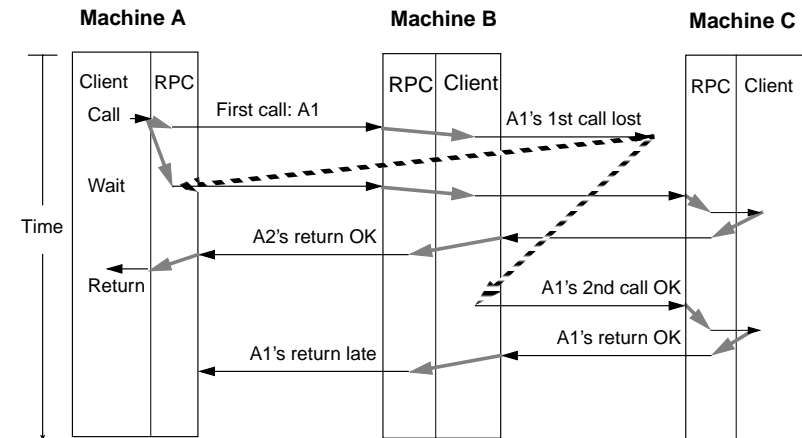
- exactly-once
- at-least-once
- last-of-many

Verwendung von
Sequenznummern



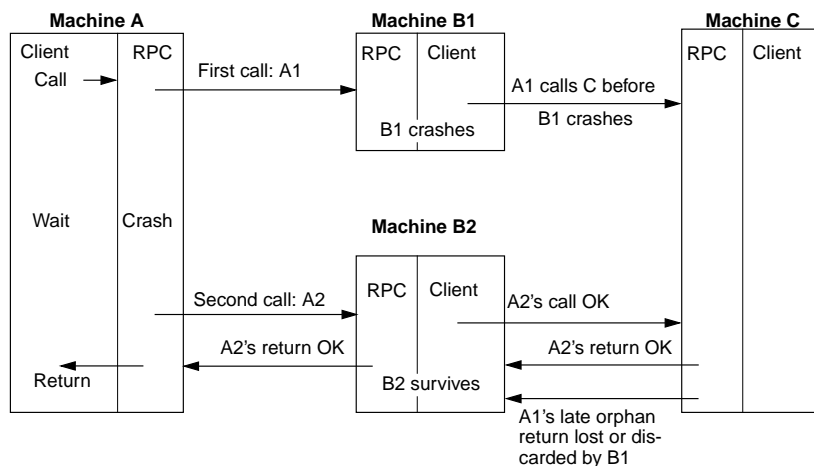
BP 1 Fernaufruf: Ideale Eigenschaften eines transparenten Mechanismus

Diese einfache Lösung ist nicht transitiv



BP 1 Fernaufruf: Ideale Eigenschaften eines transparenten Mechanismus

- Numerierung der Aufrufe löst Problem, wenn keine Zusammenbrüche auftreten.
- Gegenbeispiel im anderen Fall:



BP 1 Fernaufruf: Ideale Eigenschaften eines transparenten Mechanismus

Ausführungsschema

seriell

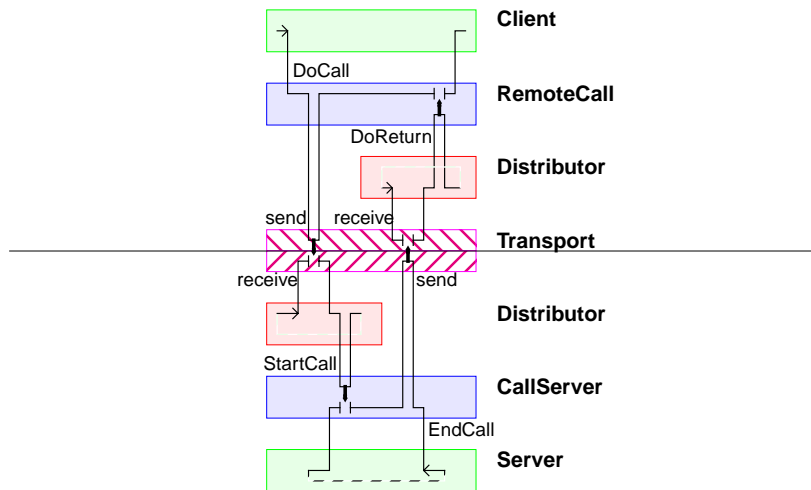
- Einfach zu realisieren durch sogenannten 'Server'-Prozeß
- Kann in nebenläufigen Systemen Verklemmungen einführen!

nebenläufig

- Bei jedem Fernaufruf neuer Prozeß auf der 'Server'-Seite
- Fester Vorrat an 'Server'-Prozessen zur Verringerung des dynamischen Aufwands
- Variabler Vorrat an 'Server'-Prozessen mit unterer und oberer Schwelle

BP 1 Fernanruf: Ideale Eigenschaften eines transparenten Mechanismus

Algorithmus von Lampson



BP 1 Fernanruf: Ideale Eigenschaften eines transparenten Mechanismus

Koordinierung durch Monitore

Konzept A (Hoare)

- **Datenstruktur mit Prozeduren (Funktionen)**
 - Daten des Monitors können nur durch die Prozeduren (Funktionen) des Monitors manipuliert werden
- **Prozeduren (Funktionen) laufen unter gegenseitigem Ausschuß**
- **zusätzlicher Datentyp condition**
 - repräsentiert Warteschlange von blockierten Prozessen
- **zugehörige Operationen**

wait	zum Blockieren und Einreihen eines Prozesses in die Warteschlange
signal	zum Deblockieren und Entfernen eines Prozesses aus der Warteschlange

Problem: Geschachtelte Monitore

BP 1 Fernanruf: Ideale Eigenschaften eines transparenten Mechanismus

Konzept B (Java)

- **Jedes Objekt besitzt**
 - einen Belegungszustand (frei, belegt)
 - eine Belegt-Warteschlange
 - eine Koordinierungswarteschlange
- **Frei wählbare Prozedurabschnitte (Funktionen) laufen unter gegenseitigem Ausschuß**
- **Belegt-Warteschlange**
 - **Einreihung**
 - bei Betreten eines koordinierten Abschnitts, falls Objekt schon belegt, sonst lediglich belegen des Objektes
 - **Ausreihung**
 - bei Verlassen eines koordinierten Abschnitts; war die Belegt-Warteschlange leer wird das Objekt frei
 - durch Aufruf von wait(); war die Belegt-Warteschlange leer wird das Objekt frei

BP 1 Fernanruf: Ideale Eigenschaften eines transparenten Mechanismus

Koordinierungs-Warteschlange

- **Einreihung**
 - durch Aufruf von wait();
- **Ausreihung**
 - eines Fadens durch die Operation notify() angewandt auf das koordinierte Objekt. War das Objekt belegt, wird der Faden in die Belegt-Warteschlange eingereiht. Ansonsten belegt er das Objekt.
 - aller Wartenden durch die Operation notifyAll() angewandt auf das koordinierte Objekt. War das Objekt belegt, werden diese Fäden in die Belegt-Warteschlange eingereiht. Ansonsten belegt einer das Objekt, die anderen werden in die Belegt-Warteschlange eingereiht
- **Lösung des Problems geschachtelter Monitore**
 - **vorzeitige Belegung**

BP 1 Fernaufruf: Ideale Eigenschaften eines transparenten Mechanismus

```
class RPC {
    public static final int CALL = 1;
    public static final int RETURN = 2;
    ...
}

class ID {
    long value;
    ...
}

class Message {
    int state; // state is CALL or RETURN
    Processor source = null, dest = null;
    ID id, request;
    Object obj; // Object whose method should be called
    Method action; // Method to be called
    Object[] val = new Object[1]; // Argument
    ...
}
```

BP 1 Fernaufruf: Ideale Eigenschaften eines transparenten Mechanismus

```
class Distributor extends Process {
    HashSet connections = new HashSet();
    // The one Process which receives and distributes messages
    public void run() {
        ResultOfReceive r;
        while (true) {
            r = receive();
            if (r.state) {
                if ((r.message.state == RPC.CALL)
                    && okToAccept(r.message)) {
                    myCallServer.startCall(r.message);
                } else {
                    if (r.message.state == RPC.RETURN) {
                        myRemoteCall.doReturn(r.message);
                    }
                }
            }
        }
    }
}
```

BP 1 Fernaufruf: Ideale Eigenschaften eines transparenten Mechanismus

```
// Make calls
class RemoteCall {
    class CallOut {
        Message m = new Message(); Condition received;
    }
    HashSet callsOut = new HashSet();
}
```

BP 1 Fernaufruf: Ideale Eigenschaften eines transparenten Mechanismus

```
public Object[] doCall(Processor d, Object o, Method a, Object[] args) {
    CallOut c = new CallOut();
    c.m.source = myMachine(); c.m.request = myMachine().uniqueID();
    c.m.dest = d; c.m.action = a; c.m.obj = o; c.m.val = args;
    c.m.state = RPC.CALL; c.received = new Condition();
    synchronized (this) {
        callsOut.add(c);
        c.m.id = myMachine().uniqueID(); c.m.dest.send(c.m);
    }
    do {
        c.received.synchronizeTimed();
        synchronized (this) {
            if (c.m.state == RPC.RETURN) break;
            c.m.id = myMachine().uniqueID(); c.m.dest.send(c.m);
        }
    } while (true);
    synchronized (this) {
        return c.m.val;
    }
}
```

BP 1 Fernaufruf: Ideale Eigenschaften eines transparenten Mechanismus

```
synchronized public void doReturn(Message m) {
    CallOut c;
    Iterator i = callsOut.iterator();
    while (i.hasNext()) {
        c = (CallOut)i.next();
        if (c.m.id == m.id) {
            c.m = m;
            callsOut.remove(c);
            synchronized (c.received) {
                c.received.notify();
            }
            return;
        }
    }
}
```

BP 1 Fernaufruf: Ideale Eigenschaften eines transparenten Mechanismus

```
class CallIn {
    Message m = new Message();
    Condition work = new Condition();
}
```

BP 1 Fernaufruf: Ideale Eigenschaften eines transparenten Mechanismus

```
// Serialize calls for each process, and assign work to worker processes
class CallServer {
    HashSet callsIn = new HashSet(), pool = new HashSet(initNumberOfWorkers);
    public void startCall(Message m) {
        CallIn w, c;
        synchronized (this) {
            Iterator i; i = callsIn.iterator();
            while (i.hasNext()) {
                c = (CallIn)i.next();
                if (c.m.request == m.request) { c.m.id = m.id; return; }
            }
            i = pool.iterator();
            // wait if the pool is empty
            while (!i.hasNext()) {
                try { wait(); } catch (InterruptedException e) { }
            }
            w = (CallIn)i.next(); pool.remove(w); callsIn.add(w); w.m = m;
        }
        w.work.signal();
    }
}
```

BP 1 Fernaufruf: Ideale Eigenschaften eines transparenten Mechanismus

```
public void endCall(CallIn w) {
    synchronized (this) {
        if (w.m.source != null) {
            w.m.source.send(w.m);
            callsIn.remove(w);
        }
        pool.add(w);
    }
    w.work.synchronize();
}
```

BP 1 Fernaufruf: Ideale Eigenschaften eines transparenten Mechanismus

```
class Worker extends Process {
    CallServer callServer = ...;

    public void run() {
        CallIn c = new CallIn();
        c.m.source = null;
        callServer.endCall(c);
        while (true) {
            try {
                c.m.val = new Object[1];
                c.m.val[0] = c.m.action.invoke(c.m.obj, null);
            } catch (IllegalAccessException e) {}
            catch (InvocationTargetException e) {}
            c.m.state = RPC.RETURN;
            callServer.endCall(c);
        }
    }
}
```

BP 1 Fernaufruf: Ideale Eigenschaften eines transparenten Mechanismus

```
class Connection {
    Processor from;
    ID lastID;
}

// The following method is part of the class Distributor

public boolean okToAccept(Message m) {
    Connection c; Iterator i;
    i = connections.iterator();
    while (i.hasNext()) {
        c = (Connection)i.next();
        if (c.from.myName.equals(m.source.myName)) {
            if (c.lastID != null) {
                if (m.id.value <= c.lastID.value) return false;
                c.lastID = m.id; return true;
            } else return false;
        }
    }
    // No record of this processor. Establish connection.
```

BP 1 Fernaufruf: Ideale Eigenschaften eines transparenten Mechanismus

```
// okToAccept continued

// No record of this processor. Establish connection.
if (m.action.getName().equals("uniqueID")) {
    return true;
    // Avoid an infinite loop; OK to duplicate this call.
}
c = new Connection();
c.from = m.source;
connections.add(c);
Connector connector = new Connector(myMachine, "connector", c);
connector.start();
return false;
}
```

BP 1 Fernaufruf: Ideale Eigenschaften eines transparenten Mechanismus

```
class Connector extends Process {
    Connection connection = ...;
    Processor myMachine = ...;
    public void run() {
        Class cl = Processor.class;
        try {
            connection.lastID
                = (ID)((myMachine.remoteCall
                    .doCall(connection.from,
                        connection.from,
                        cl.getMethod("uniqueID", null), null))[0])
        } catch (NoSuchMethodException e) {...}
    }
}
```

BP 1 Fernaufruf: Ideale Eigenschaften eines transparenten Mechanismus

Übertragung	keine Fehler	keine Fehler erhöhte Verzögerung	Paketverlust	Paketverlust	Paketverlust	Paketverlust
Prozessoren				Server- ausfall	Clientausfall	Client- und Serverausfall
Maybe (No Recovery)	OP = 1 Res = 1	OP = 1 RES ≤ 1	OP ≤ 1 RES = 0	OP ≤ 1 RES = 0	OP ≤ 1 RES = 0	OP ≤ 1 RES = 0
at least once (Regular Server)	OP = 1 Res = 1	OP ≥ 1 RES ≥ 1	OP ≥ 1 RES ≥ 1	OP ≥ 0 RES ≥ 0	OP ≥ 0 RES = 0	OP ≥ 0 RES = 0
at least once (Recov. Server)	OP = 1 Res = 1	OP ≥ 1 RES ≥ 1	OP ≥ 1 RES ≥ 1	OP ≥ 1 RES ≥ 1	OP ≥ 0 RES = 0	OP ≥ 0 RES = 0
last of many (Regular Server)	OP = 1 Res = 1	OP ≥ 1 RES ≤ 1	OP ≥ 1 Res = 1	OP ≥ 0 RES ≤ 1	OP ≥ 0 RES = 0	OP ≥ 0 RES = 0
last of many (Recov. Server)	OP = 1 Res = 1	OP ≥ 1 RES ≥ 1	OP ≥ 1 Res = 1	OP ≥ 1 Res = 1	OP ≥ 0 RES = 0	OP ≥ 0 RES = 0
last one (Recov. C & S)	OP = 1 Res = 1	OP = 1 Res = 1	OP = 1 Res = 1	OP = 1 Res = 1	OP ≥ 1 Res = 1	OP ≥ 1 Res = 1
only once t1 (Regular Server)	OP = 1 Res = 1	OP = 1 Res = 1	OP = 1 Res = 1	OP ≤ 1 RES = 0	OP ≤ 1 RES = 0	OP ≤ 1 RES = 0
only once t2 (Recov. C & S)	OP = 1 Res = 1	OP = 1 Res = 1	OP = 1 Res = 1	OP = 1 Res = 1	OP = 1 Res = 1	OP = 1 Res = 1
all or nothing (Transactions)	OP = 1 Res = 1	OP = 1 Res = 1	OP = 1 Res = 1	OP = 1 Res = 1	OP = 0 RES = 0	OP = 0 RES = 0

30.10.01 Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann
Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

3.3-29

BP 1 Fernaufruf: Ideale Eigenschaften eines transparenten Mechanismus

Verwaiste Aufrufe (orphans)

- Fragestellung
- Extermination
 - gezielte Elimination
 - Adoption
- Expiration
 - Zu kurze Intervalle verursachen auch bei Fehlerfreiheit Abweichen von der exactly-once-Semantik
 - Problem: Uhrensynchronisation, um Verweilzeit im Kommunikationssystem berücksichtigen zu können
- Deadlining with postponement
- Reincarnation
 - weak last-one
 - gentle

30.10.01 Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann
Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

3.3-30

BP 1 Fernaufruf: Ideale Eigenschaften eines transparenten Mechanismus

Weitere Punkte

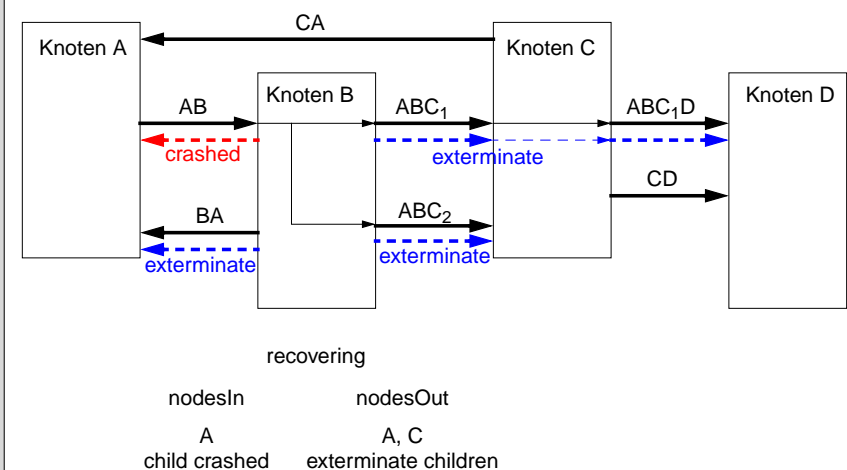
- wrap around
 - 16 Bit, jede msec eine Nachricht: 1,1 min
 - 32 Bit, jede msec eine Nachricht: 1200 Stunden
- Stabilität der Sequenznummern bei Prozessorausfällen
 - Generation zusammen mit Sequenznummer;
 - Generation nur geändert bei Recovery

30.10.01 Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann
Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

3.3-31

BP 1 Fernaufruf: Ideale Eigenschaften eines transparenten Mechanismus

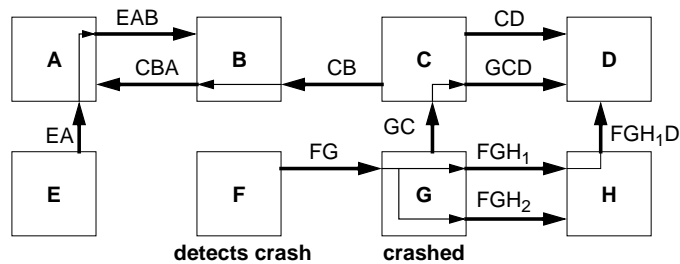
Extermination



30.10.01 Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann
Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig

3.3-32

BP 1 Fernaufruf: Ideale Eigenschaften eines transparenten Mechanismus



Extermination: GC, FGH₁, FGH₂ rekursiv: GCD, FGH₁D crashed: FG

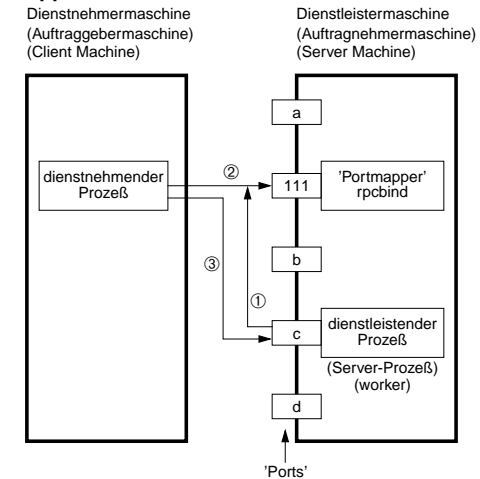
Expiration: GC, GCD, FGH₁, FGH₂, FGH₁D

Reincarnation: regular: alles
gentle: wie bei Expiration

BP 1 Fernaufruf: SUN RPC

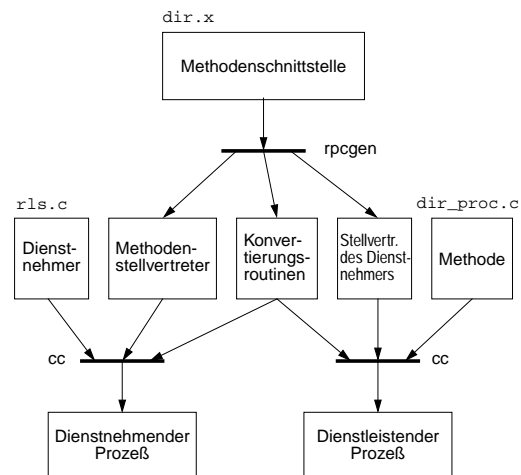
3.4 Der SUN RPC (unter Solaris)

3.4.1 Ports und Portmapper



BP 1 Fernaufruf: Generierung von Client und Server

3.4.2 Generierung von Client und Server am Beispiel eines Fernaufrufs zur Ermittlung des Inhalts eines Katalogs



BP 1 Fernaufruf: Generierung von Client und Server

◆ Erzeugung der Marshaling-Methoden

```
faii45a% rpcgen dir.x
dir.x  Beschreibung der Methodenschnittstelle und der Einbettung
       in den Server
Erzeugt verschiedene Hilfsdateien
dir.h   gemeinsame Typdefinitionen
dir_svc.c  Rahmenprogramm des Auftragnehmers
dir_xdr.c  Prozeduren zur Konvertierung (Rückkonvertierung)
           in (aus) der Datendarstellung, die bei der
           Übertragung von Parametern verwendet wird.
dir_clnt.c Stellvertreter der Methode.
```

◆ Vorübersetzung von dir_xdr.c

```
faii45a% cc -c dir_xdr.c
```

BP 1	Fernauf: Generierung von Client und Server
	<ul style="list-style-type: none"> ◆ Erzeugung des Auftraggebers (Client) rls.c faui45a% cc rls.c dir_clnt.c dir_xdr.o -o rls -lnsl ◆ Erzeugung des Auftragnehmers (Server) mit der Methode dir_proc.c faui45a% cc dir_svc.c dir_proc.c dir_xdr.o -o dir_svc -lnsl ◆ Ausführung <ul style="list-style-type: none"> • Start des Dienstes (Server): dir_svc& ◆ Start des Dienstnutzers: (Client) rls faui45a ~
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.4-37

BP 1	Fernauf: Die beteiligten Klassen
3.4.3	Die beteiligten Klassen <ul style="list-style-type: none"> □ Datentypen im unverteilten Fall <pre> // Datenstrukturen, die in der Methodenschnittstelle benutzt werden typedef char *nametype; // Typ der Dateinamen typedef struct namenode *namelist; // Typ von Verweisen auf Elemente // der Namensliste struct namenode { // Typ der Listenelemente; // sie bestehen aus: nametype name; // einem Dateinamen name und namelist next; // einem Verweis next auf das // naechste Listenelement. }; struct readdir_res { // Typ des Methodenergebnisses, // bestehend aus: int errno; // einer Fehlerkennung und, // falls sie den Wert 0 besitzt, namelist list; // einer Liste von Dateinamen. }; </pre>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.4-38

BP 1	Fernauf: Die beteiligten Klassen
	<ul style="list-style-type: none"> □ Datentypen im verteilten Fall (Fernaufrufe) <pre> const MAXNAMELENGTH = 255; // Typ der Dateinamen typedef string nametype<MAXNAMELENGTH>; // Typ von Verweisen auf Elemente der Namensliste typedef struct namenode *namelist; // Typ der Listenelemente; sie bestehen aus einem Dateinamen name und einem // Verweis next auf das nächste Listenelement. struct namenode { nametype name; namelist next; }; </pre>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.4-39

BP 1	Fernauf: Die beteiligten Klassen
◆	Typ des Methodenergebnisses, bestehend aus einer Fehlerkennung und (falls sie den Wert 0 hat), einer Liste von Dateinamen. <p>Anmerkung: Bei union wird in den Programmen eine Struktur mit einer zusätzlichen Komponente gebildet. Die Union-Komponente wird durch den Union-Typnamen mit angeschlossener Zeichenfolge <code>_u</code> identifiziert.</p> <pre> union readdir_res switch(int errno) { case 0: namelist list; default: void; }; </pre>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.4-40

3.4.4 Die Schnittstellenbeschreibung

```

program DIRPROG {
// Der Dienstleister (Server) erhält den Namen DIRPROG

    version DIRVERS {
        readdir_res READDIR(nametype) = 1;
        // Die Version DIRVERS stellt die Methode READDIR zur
        // Verfügung, die unter der Methodennummer 1 erreicht wird.
        // Dienstnehmer rufen diese Methode mit dem Namen readdir_1 auf
        // (die Zahl nach dem Unterstrich ist die Versionsnummer).

    } = 1;
    // Die Version DIRVERS wird unter der Kennung 1 registriert.
} = 0x20000076;
// Der Dienstleister (Server) wird unter der Nummer 0x20000076
// registriert.

```

XDR sieht ein Muster zur Konvertierung zwischen C-Datentypen und einer externen Darstellung als Bitfolge vor. Für die Standardtypen von C sind Bibliotheksroutinen verfügbar. Diese Routinen zusammen mit unterstützenden Routinen bilden die Grundlage, auf der für jeden benutzerdefinierten Datentyp Routinen implementiert werden können, die die Konvertierung in beiden Richtungen vornehmen. Für jeden Datentyp wird eine Routine benötigt mit zwei Argumenten:

```

bool_t
xdr_<type>(xdrs, argresp)
    XDR *xdrs;           // Verweis auf ein XDR-Objekt, in das bzw.
                        // aus dem konvertiert werden soll.
                        // Das XDR-Objekt enthält eine Komponente,
                        // die angibt, ob nach XDR konvertiert
                        // (ENCODE) werden soll oder
                        // aus XDR (DECODE)
    <type> *argresp;

```

Struktur eines XDR-Objektes

```

enum xdr_op {XDR_ENCODE = 0, XDR_DECODE = 1, XDR_FREE = 2 };

typedef struct {
    enum xdr_op x_op; /* operation; fast additional param */
    struct xdr_ops {
        bool_t (*x_getlong)(); /* get a long from stream */
        bool_t (*x_putlong)(); /* put a long to stream */
        bool_t (*x_getbytes)(); /* get bytes from stream */
        bool_t (*x_putbytes)(); /* put bytes to stream */
        u_int (*x_getpostn)(); /* return stream offset */
        bool_t (*x_setpostn)(); /* reposition offset */
        long * (*x_inline)(); /* ptr to buffered data */
        void (*x_destroy)(); /* free private area */
    } *x_ops;
    caddr_t x_public; /* users' data */
    caddr_t x_private; /* pointer to private data */
    caddr_t x_base; /* private for position info */
    int x_handy; /* extra private word */
} XDR;

```

Aus den Datenstrukturen für rls werden unter Rückgriff auf die Konversionsroutinen für Standarddatentypen folgende Routinen erzeugt;

```

/*
 * Please do not edit this file.
 * It was generated using rpcgen.
 */
#include "dir.h"
bool_t
xdr_nametype(xdrs, objp)
    register XDR *xdrs;
    nametype *objp;
{
    register long *buf;
    if (!xdr_string(xdrs, objp, MAXNAMELENGTH))
        return (FALSE);
    return (TRUE);
}

```

BP 1	Fernauf: Schnittstellenbeschreibung und External Data Representation (XDR)
	<pre> bool_t xdr_namelist(xdrs, objp) { register XDR *xdrs; namelist *objp; register long *buf; if (!xdr_pointer(xdrs, (char **)objp, sizeof (struct namenode), (xdrproc_t) xdr_namenode)) return (FALSE); return (TRUE); } </pre>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.4-45

BP 1	Fernauf: Schnittstellenbeschreibung und External Data Representation (XDR)
	<pre> bool_t xdr_namenode(xdrs, objp) { register XDR *xdrs; namenode *objp; register long *buf; if (!xdr_nametype(xdrs, &objp->name)) return (FALSE); if (!xdr_namelist(xdrs, &objp->next)) return (FALSE); return (TRUE); } </pre>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.4-46

BP 1	Fernauf: Schnittstellenbeschreibung und External Data Representation (XDR)
	<pre> bool_t xdr_readdir_res(xdrs, objp) { register XDR *xdrs; readdir_res *objp; register long *buf; if (!xdr_int(xdrs, &objp->errno)) return (FALSE); switch (objp->errno) { case 0: if (!xdr_namelist(xdrs, &objp->readdir_res_u.list)) return (FALSE); break; } return (TRUE); } </pre>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.4-47

BP 1	Fernauf: Die dienstleistende Routine (dir_proc.c)
3.4.5 Die verschiedenen Routinen <input type="checkbox"/> Die dienstleistende Routine (dir_proc.c)	<pre> readdir_res * readdir_1(dirname, req) nametype *dirname; struct svc_req *req; // Der Methodenname besteht aus einer symbolischen Bezeichnung // gefolgt von einem Unterstrich, auf den die Versionsnummer folgt. // Parameter- und Ergebnistypen werden in der Methodenschnittstelle // definiert. { DIR *dirp; struct direct *d; namelist *nlp, nl; static readdir_res res; </pre>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.4-48

BP 1	Fernauf: Die dienstleistende Routine (dir_proc.c)
	<pre> dirp = opendir(*dirname); if(dirp == NULL) { res.errno = errno; return(&res); } xdr_free(xdr_readdir_res, (char *)&res); nlp = &res.readdir_res_u.list; printf("\n"); while (d = readdir(dirp)) { nl = *nlp = (namenode *) malloc(sizeof(namenode)); nl->name = strdup(d->d_name); nlp = &nl->next; } *nlp = NULL; </pre>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.4-49

BP 1	Fernauf: Die dienstleistende Routine (dir_proc.c)
	<pre> res.errno = 0; closedir(dirp); printf("\n"); return(&res); } </pre>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.4-50

BP 1	Fernauf: Die dienstleistende Routine (dir_proc.c)
<p>♦ Zur Unterstützung der Programmierung steht eine Bibliothek spezieller Routinen zu folgenden Themen zur Verfügung:</p> <ul style="list-style-type: none"> • Erzeugung und Manipulation von Identifikatoren für den Dienstzugang rpc_clnt_create, 14 Funktionen • Funktionsaufrufe durch den Dienstnehmer rpc_clnt_calls, 10 Funktionen • Asynchrone Fernaufrufe rpc_rac, 4 Funktionen • Erzeugung von Identifikatoren für Dienstleister rpc_svc_create, 9 Funktionen • Organisation der Dienstleister rpc_svc_calls, 13 Funktionen 	
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.4-51

BP 1	Fernauf: Die dienstleistende Routine (dir_proc.c)
	<ul style="list-style-type: none"> • Registrierung von Dienstleistern rpc_svc_reg, 6 Funktionen • Behandlung von Fehlern beim Dienstleister rpc_svc_err, 7 Funktionen • Unterstützung der Konvertierung nach und von XDR rpc_xdr, 7 Funktionen • Authentisierung des Dienstnehmers rpc_clnt_auth, 4 Funktionen • Sicherung des Datentransports secure_rpc, 12 Funktionen • Authentisierung mit dem Kerberos-Protokoll kerberos_rpc, 3 Funktionen
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.4-52

BP 1	Fernauf: Der Dienstnehmer (rls.c)
	<p>◆ Der Dienstnehmer (rls.c)</p> <pre> main(argc, argv) int argc; char *argv[]; // Das Programm wird mit zwei Parametern gestartet, nämlich dem // Namen des Prozessors, auf dem das auszugebende Adreßbuch liegt // und dem Pfadnamen des auszugebenden Adreßbuchs. { CLIENT *cl; char *server, *dir; readdir_res *result; // In der Schnittstellenbeschreibung namelist nl; // definierte Typen server = argv[1]; dir = argv[2]; // Übernahme der Parameter. </pre>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.4-53

BP 1	Fernauf: Der Dienstnehmer (rls.c)
	<pre> cl = clnt_create(server, DIRPROG, DIRVERS, "tcp"); // Erzeugung eines Deskriptors für den Serverzugang result = readdir_1(&dir, cl); // Aufruf der Methode readdir mit Versionsnummer 1 in dem Server, // der durch den Deskriptor cl identifiziert wird. for (nl = result->readdir_res_u.list; nl != NULL; nl = nl->next) { printf("%s\n", nl->name); } // Ausgabe des Ergebnisses aus dem Methodenaufruf exit(0); } </pre>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.4-54

BP 1	Fernauf: Stellvertreter des Dienstleisters (generiert)
	<p>◆ Der Stellvertreter des Dienstleisters (generiert)</p> <pre> // Default timeout can be changed using clnt_control() static struct timeval TIMEOUT = { 25, 0 }; readdir_res * readdir_1(argp, clnt) nametype *argp; CLIENT *clnt; { static readdir_res clnt_res; memset((char *)&clnt_res, 0, sizeof (clnt_res)); if (clnt_call(clnt, READDIR, (xdrproc_t) xdr_nametype, (caddr_t) argp, (xdrproc_t) xdr_readdir_res, (caddr_t) &clnt_res, TIMEOUT) != RPC_SUCCESS) { return (NULL); } return (&clnt_res); } </pre>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.4-55

BP 1	Fernauf: Stellvertreter des Dienstnehmers (generiert)
	<p>◆ Der Stellvertreter des Dienstnehmers (generiert)</p> <pre> main() { register SVCXPRT *transp; struct netconfig *nconf = NULL; // Deregisters all mappings between the triple // [DIRPROG, DIRVERS, *] and ports on the local machine's // triple [DIRPROG, DIRVERS, *] and ports on the local machine's // portmap service. transp = svc_tli_create(0, nconf, NULL, 0, 0) // Creates an RPC server handle, and returns a pointer to it. svc_reg(transp, DIRPROG, DIRVERS, dirprog_1, 0); // Associates DIRPROG and DIRVERS with the service dispatch // procedure, dispatch. A mapping of the triple // [DIRPROG, DIRVERS, IPPROTO_UDP] to transp is established with // the local portmap service. } </pre>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.4-56

BP 1	Fernauf: Stellvertreter des Dienstnehmers (generiert)
	<pre> svc_run(); // Normally, this routine only returns in the case of some errors. // It waits for RPC requests to arrive, and calls the appropriate // service procedure using svc_getreq() when one arrives. </pre>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.4-57

BP 1	Fernauf: Stellvertreter des Dienstnehmers (generiert)
<div>□</div> <div>◆</div>	<p>Beschreibung der im Beispiel vom Dienstleister benutzten Routinen</p> <pre> SVCXPRT *svc_tli_create(const int fildes, const struct netconfig *netconf, const struct t_bind *bindaddr, const u_int sendsz, const u_int recvsz); </pre> <p>This routine creates an RPC server handle, and returns a pointer to it. fildes is the file descriptor on which the service is listening. If fildes is RPC_ANYFD, it opens a file descriptor on the transport specified by netconf. If the file descriptor is unbound and bindaddr is non-null fildes is bound to the address specified by bindaddr, otherwise fildes is bound to a default address chosen by the transport. In the case where the default address is chosen, the number of outstanding connect requests is set to 8 for connection-oriented transports. The user may specify the size of the send and receive buffers with the parameters sendsz and recvsz ; values of 0 choose suitable defaults. This routine returns NULL if it fails, and an error message is logged. The server is not registered with the rpcbind(1M) service.</p>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.4-58

BP 1	Fernauf: Stellvertreter des Dienstnehmers (generiert)
◆	<pre> int svc_reg(const SVCXPRT *xp, const u_long prognum, const u_long versnum, const void (*dispatch), const struct netconfig *netconf); </pre> <p>Associates prognum and versnum with the service dispatch procedure, dispatch. If netconf is NULL, the service is not registered with the rpcbind service. For example, if a service has already been registered using some other means, such as inetd (see inetd(1M)), it will not need to be registered again. If netconf is non-zero, then a mapping of the triple [prognum, versnum, netconf->nc_netid] to xp->xp_laddr is established with the local rpcbind service.</p> <p>The svc_reg() routine returns 1 if it succeeds, and 0 otherwise.</p>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.4-59

BP 1	Fernauf: Stellvertreter des Dienstnehmers (generiert)
◆	<pre> void svc_run(void); </pre> <p>This routine never returns. In single threaded mode, it waits for RPC requests to arrive, and calls the appropriate service procedure using svc_getreq_poll() when one arrives. This procedure is usually waiting for the poll(2) library call to return.</p> <p>Applications executing in the Automatic or User MT modes should invoke this function exactly once. In the Automatic MT mode, it will create threads to service client requests. In the User MT mode, it will provide a framework for service developers to create and manage their own threads for servicing client requests.</p>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.4-60

BP 1 Fernaufruf: Stellvertreter des Dienstnehmers (generiert)

```
❑ Der Stellvertreter des Dienstnehmers (generiert)
static void
dirprog_1(rqstp, transp)
    struct svc_req *rqstp;
    // Verweis auf Auftragsblock, der Programmnummer, Versionsnummer,
    // Prozedurnummer und Sicherheitsinformationen enthält
    register SVCXPRT *transp;
    // Transportobjekt

{ union {
    nametype readdir_1_arg;
} argument;
char *result;
bool_t (*xdr_argument)(), (*xdr_result)();
char *(*local)();
```

BP 1 Fernaufruf: Stellvertreter des Dienstnehmers (generiert)

```
switch (rqstp->rq_proc) {
case NULLPROC:
    // Returns with no results. Can be used as a simple test for
    // detecting if a remote programm is running.
    (void) svc_sendreply(transp, xdr_void, (char *)NULL);
    return;

case READDIR:
    // Funktion readdir ist auszuführen
    xdr_argument = xdr_nametype;
    xdr_result = xdr_readdir_res;
    local = (char *(*())()) readdir_1;
    break;

default:
    svcerr_noproc(transp);
    return;
}
```

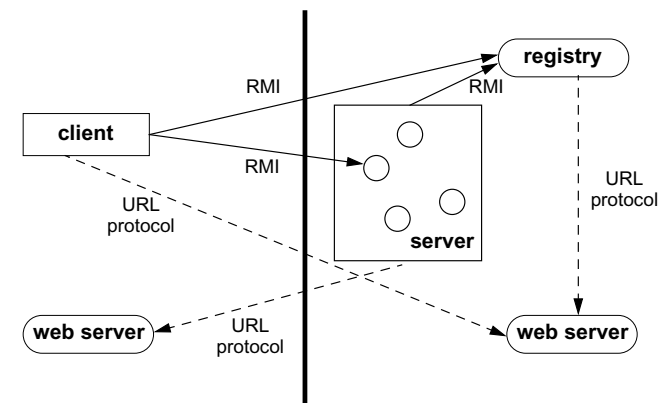
BP 1 Fernaufruf: Stellvertreter des Dienstnehmers (generiert)

```
bzero((char *)&argument, sizeof(argument));
if (!svc_getargs(transp, xdr_argument, &argument)) {
    // Argument konvertieren
    svcerr_decode(transp); return;
}
result = (*local)(&argument, rqstp);
// Gewünschte Funktion ausführen
if (result != NULL && !svc_sendreply(transp, xdr_result, result)) {
    // Ergebnis zurücksenden
    svcerr_systemerr(transp);
}
if (!svc_freeargs(transp, xdr_argument, &argument)) {
    fprintf(stderr, "unable to free arguments");
    exit(1);
}
return;
}
```

BP 1 Fernaufruf: Java Remote Method Invocation

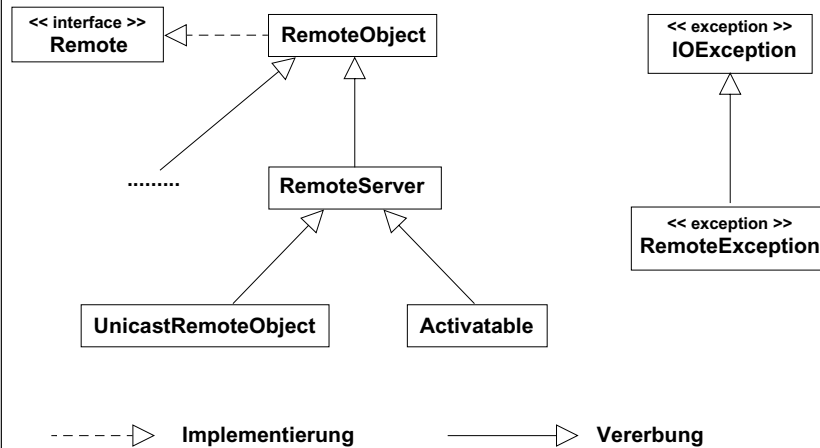
3.5 Java Remote Method Invocation (RMI)

❑ Gesamtstruktur



BP 1 Fernaufruf: Java Remote Method Invocation

Klassen und Schnittstellen



BP 1 Fernaufruf: Java Remote Method Invocation

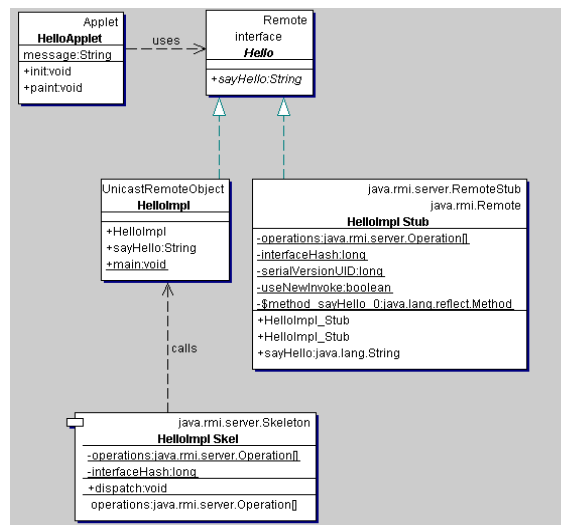
Vorgehensweise

1. Definition der fernaufrufbaren Funktionen als Java-Interface
2. Schreiben der Klassen des Dienstleisters
3. Schreiben des Dienstnehmers (als Applet)

Beispiel: Hello World

- Der Dienstleister (server) `HelloImpl` gibt nach Aufruf der Methode `sayHello()` die Zeichenkette "Hello World" zurück.
- Der Dienstnehmer `HelloApplet` ruft mittels Fernaufruf die Methode `sayHello()` auf und gibt die zurückerhaltene Zeichenkette aus.

BP 1 Fernaufruf: Java Remote Method Invocation



BP 1 Fernaufruf: Java Remote Method Invocation

1. Definition der fernaufrufbaren Funktionen als Java-Interface

```

package examples.hello;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Hello extends Remote {
    String sayHello() throws RemoteException;
}
  
```

- The remote interface must be declared public. Otherwise, unless a client is in the same package as the remote interface, the client will get an error when attempting to load a remote object that implements the remote interface.
- The remote interface extends the `java.rmi.Remote` interface.
- Each method must declare `java.rmi.RemoteException` (or a superclass of `RemoteException`) in its throws clause, in addition to any application-specific exceptions.
- The data type of any remote object that is passed as an argument or return value (either directly or embedded within a local object) must be declared as the remote interface type (for example, `Hello`) not the implementation class (`HelloImpl`).

BP 1 Fernaufruf: Java Remote Method Invocation**2. Schreiben der Klassen des Dienstleisters**

```
package examples.hello;

import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.rmi.server.UnicastRemoteObject;

public class HelloImpl extends UnicastRemoteObject
    implements Hello {

    public HelloImpl() throws RemoteException {
        super();
    }

    public String sayHello() {
        return "Hello World!";
    }
}
```

BP 1 Fernaufruf: Java Remote Method Invocation

```
public static void main(String args[]) {

    // Create and install a security manager
    if (System.getSecurityManager() == null) {
        System.setSecurityManager(new RMISecurityManager());
    }
    try {
        HelloImpl obj = new HelloImpl();
        // Bind this object instance to the name "HelloServer"
        Naming.rebind("///HelloServer", obj);
    } catch (Exception e) {
        System.out.println("New HelloImpl err: "
            + e.getMessage());
        e.printStackTrace();
    }
}
```

BP 1 Fernaufruf: Java Remote Method Invocation**3. Schreiben des Dienstnehmers (als Applet)**

```
package examples.hello;

import java.applet.Applet;
import java.awt.Graphics;
import java.rmi.Naming;
import java.rmi.RemoteException;

public class HelloApplet extends Applet {
    String message = "blank";

    // "obj" is the identifier that we'll use to refer
    // to the remote object that implements the "Hello"
    // interface
    Hello obj = null;
}
```

BP 1 Fernaufruf: Java Remote Method Invocation

```
public void init() {
    try {
        obj = (Hello)Naming.lookup("///"
            + getCodeBase().getHost()
            + "/HelloServer");

        message = obj.sayHello();
    } catch (Exception e) {
        System.out.println("HelloApplet exception: "
            + e.getMessage());
        e.printStackTrace();
    }
}

public void paint(Graphics g) {
    g.drawString(message, 25, 50);
}
}
```

BP 1 Fernaufruf: Java Remote Method Invocation

- Systemerzeugung
 - Kompilieren der drei Klassen
 - Mittels rmic Stub und Skeleton erzeugen
 - Bestandteile verteilen
- Am dienstleistenden Rechner
 - Registratur starten
 - Dienstleister (server) starten
- Am dienstnehmenden Rechner
 - Applet starten

BP 1 Fernaufruf: Stub class generated by rmic

```
// Stub class generated by rmic, do not edit.
// Contents subject to change without notice.

package examples.hello;

public final class HelloImpl_Stub
    extends java.rmi.server.RemoteStub
    implements examples.hello.Hello, java.rmi.Remote
{
    private static final java.rmi.server.Operation[] operations = {
        new java.rmi.server.Operation("java.lang.String sayHello()")
    };

    private static final long interfaceHash = 6486744599627128933L;

    private static final long serialVersionUID = 2;

    private static boolean useNewInvoke;
    private static java.lang.reflect.Method $method_sayHello_0;
```

BP 1 Fernaufruf: Stub class generated by rmic

```
static {
    try {
        java.rmi.server.RemoteRef.class.getMethod("invoke",
            new java.lang.Class[] {
                java.rmi.Remote.class,
                java.lang.reflect.Method.class,
                java.lang.Object[].class,
                long.class
            });
        useNewInvoke = true;
        $method_sayHello_0 =
            examples.hello.Hello.class.getMethod("sayHello",
                new java.lang.Class[] {});
    } catch (java.lang.NoSuchMethodException e) {
        useNewInvoke = false;
    }
}
```

BP 1 Fernaufruf: Stub class generated by rmic

```
// constructors
public HelloImpl_Stub() {
    super();
}

public HelloImpl_Stub(java.rmi.server.RemoteRef ref) {
    super(ref);
}
```

BP 1	Fernauf: Stub class generated by rmic
	<pre>// methods from remote interfaces // implementation of sayHello() public java.lang.String sayHello() throws java.rmi.RemoteException { try { if (useNewInvoke) { Object \$result = ref.invoke(this, \$method_sayHello_0, null, 6043973830760146143L); return ((java.lang.String) \$result); } else { java.rmi.server.RemoteCall call = ref.newCall((java.rmi.server.RemoteObject) this, operations, 0, interfaceHash); ref.invoke(call); java.lang.String \$result; try { java.io.ObjectInput in = call.getInputStream(); \$result = (java.lang.String) in.readObject(); } } } }</pre>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.5-77

BP 1	Fernauf: Stub class generated by rmic
	<pre> } catch (java.io.IOException e) { throw new java.rmi .UnmarshalException ("error unmarshalling return", e); } catch (java.lang.ClassNotFoundException e) { throw new java.rmi .UnmarshalException ("error unmarshalling return", e); } finally { ref.done(call); } return \$result; } } catch (java.lang.RuntimeException e) { throw e; } catch (java.rmi.RemoteException e) { throw e; } catch (java.lang.Exception e) { throw new java.rmi .UnexpectedException("undeclared checked exception", e); } }</pre>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.5-78

BP 1	Fernauf: Skeleton class generated by rmic
	<pre>// Skeleton class generated by rmic, do not edit. // Contents subject to change without notice. package examples.hello; public final class HelloImpl_Skel implements java.rmi.server.Skeleton { private static final java.rmi.server.Operation[] operations = { new java.rmi.server.Operation("java.lang.String sayHello()") }; private static final long interfaceHash = 6486744599627128933L; public java.rmi.server.Operation[] getOperations() { return (java.rmi.server.Operation[]) operations.clone(); } }</pre>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.5-79

BP 1	Fernauf: Skeleton class generated by rmic
	<pre>public void dispatch(java.rmi.Remote obj, java.rmi.server.RemoteCall call, int opnum, long hash) throws java.lang.Exception { if (opnum < 0) { if (hash == 6043973830760146143L) { opnum = 0; } else { throw new java.rmi .UnmarshalException("invalid method hash"); } } else { if (hash != interfaceHash) throw new java.rmi.server .SkeletonMismatchException ("interface hash mismatch"); } examples.hello.HelloImpl server = (examples.hello.HelloImpl) obj; }</pre>
30.10.01	<small>Universität Erlangen-Nürnberg, Informatik 4, F. Hofmann Reproduktion jeder Art oder Verwendung dieser Unterlage zu Lehrzwecken außerhalb der Universität Erlangen-Nürnberg ist ohne Genehmigung des Autors unzulässig</small> 3.5-80

```
switch (opnum) {
  case 0: // sayHello()
    { call.releaseInputStream();
      java.lang.String $result = server.sayHello();
      try {
        java.io.ObjectOutput out
          = call.getResultStream(true);
        out.writeObject($result);
      } catch (java.io.IOException e) {
        throw new java.rmi.MarshalException
          ("error marshalling return", e);
      }
      break;
    }
  default:
    throw new java.rmi
      .UnmarshalException("invalid method number");
}
```