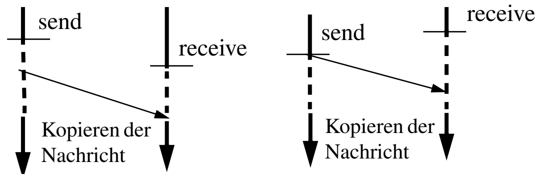
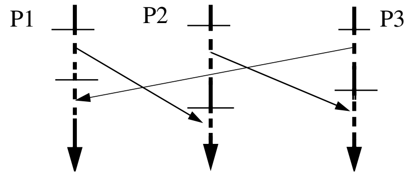
3.2 Nachrichtenaustausch

✎ Synchroner Nachrichtenaustausch

Ein Nachrichtenaustausch ist <u>synchron</u> (*synchronous*), wenn Sender und Empfänger blockierende Sende- und Empfangensoperationen verwenden.



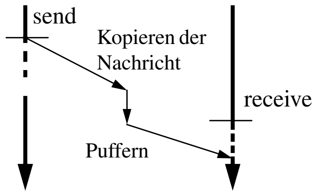
➯ Verklemmung



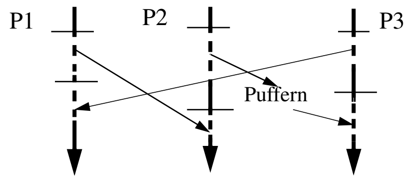
✎ Asynchroner Nachrichtenaustausch

Ein Nachrichtenaustausch ist <u>asynchron</u> (*asynchronous*), wenn mindestens einer der an der Kommunikation beteiligten Prozesse eine nicht blockierende Sende- und Empfangensoperation verwenden.

Der sendende Prozeß wird im Falle eines vollen Nachrichtenpuffers so lange blockiert (<u>pufferblockierendes Senden</u>), bis wieder Speicher frei ist.



➯ keine Verklemmung



Client1:	Client2:	
...	...	
outs1.write(...);	outs2.write(...);	//need service 1
outs2.write(...);	outs1.write(...);	//need service 2
...	...	
ins2.read(...);	ins1.read(...);	
ins1.read(...);	ins2.read(...);	
...	...	
Server1:	Service2:	
...	...	
ins.read(...);	ins.read(...);	
...	...	
outs.write(...);	outs.write(...);	
...	...	

➯ doch Verklemmung

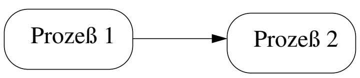
```
Client1:
...
outs1.write(...);
...
ins1.read(...);
outs2.write(...);
...

Server:
...
ins1.read(...);
ins2.read(...);
...                                   //service
outs1.write(...);
...
```

✎ Kommunikationskanäle

Indem man Kommunikationskanäle durch Sende- und Empfangensoperationen bezeichnet, legt man für eine Nachricht eine <u>Adresse</u> fest.

• direkte Benennung

Bei <u>direkter Benennung</u> (*direct naming*) werden die Prozeßnamen zur Bezeichnung von Sender und Empfänger verwendet.



➯ Pipeline im UNIX-Betriebssystem

<u>Probleme</u>

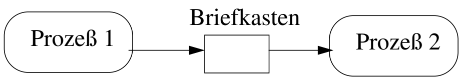
1. fehlende Strukturierung von Nachrichtenströmen zwischen zwei Prozessen
2. das notwendige Festschreiben der Namen späterer Kommunikationspartner
3. die ungünstige Formulierung von Client/Server-Systemen

➯ falls mehrere Drucker vorhanden sind

• Briefkästen

Ein <u>Briefkasten</u> (*mailbox*) ist ein mehreren Prozessen namentlich bekannter, globaler Speicherplatz, in den diese Prozesse Nachrichten durch Senden ablegen und durch Empfangen wieder herausnehmen können.

.



➯ Client/Server Systeme

<u>Probleme</u>

1. Aushungern einzelner Sender
2. Mehrfaches Empfangen
3. Einrichten eines dritten Datenbereichs

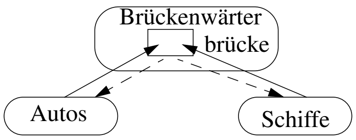
• Port

Einen Kommunikationskanal, dessen Name wie der eines Briefkastens global bekannt ist, der jedoch an einen bestimmten Prozeß gebunden ist, bezeichnet man als <u>Port</u>.

- Ein Port kann anders als ein Briefkasten nur in einer Kommunikationsrichtung benutzt werden.

➪
```
P: process
    export port1, port2;
    port1 : send port (nachrichtentyp1);
    port2 : receive port (nachrichtentyp2);
    ...
    send nachricht to port1;
    receive nachricht from port2;
    ...
end process
```

➪ In Holland sieht man oft Zugbrücken, die kleine Kanäle überspannen. Ist die Brücke heruntergeklappt, können Autos über sie fahren. Damit ein Schiff die Brücke passieren kann, muß sie aufgeklappt werden.



- Ein Auftragnehmer liefert in der Regel seinem Klienten nach Erbringen der Dienstleistung eine <u>Rückmeldung</u> (*reply*).

➪
```
send nachricht to port return kennung
await rückmeldung from port with kennung
```

- Ports kombinieren die Vorteile von Briefkästen mit der einfachen Implementierbarkeit von Pipelines.

<u>Probleme</u>

1. Synchronisation beim Zugriff auf Ports
2. nicht universell verwendbar

• Verbindung (*Connection*)

Wenn zwei Kommunikationspartner für einen Kanal jeweils einen eigenen Namen wählen, muß zusätzlich noch festgelegt werden, daß beide Namen den gleichen Kanal bezeichnen. Dies bezeichnet man als Einrichten einer <u>Verbindung</u>.

.

➪ in Java

DatagramSocket:	direkte Benennung von Ports
Socket:	Verbindung

4 Abstraktion von Nachrichtenaustausch

Bei einem <u>datenorientierten</u> Nachrichtenaustausch werden Daten von einem Prozeß gesendet und von einem anderen empfangen. Eine Kommunikation, bei der die Aktivitäten im Vordergrund stehen, wird <u>aktionsorientiert</u> genannt.

4.1 Prozedurfernaufrufe (RPC)

✎ Prozeduren als Kommunikationsmittel

- <u>Prozedurfernaufruf</u> (*remote procedure call, RPC*): Ein Prozeß ruft eine von einem anderen Prozeß bereitgestellte Prozedur auf, die in einem anderen Datenbereich arbeitet.

➪
```
auftraggeber: process
- Datenaustausch
    eingabe : ...
    ausgabe : ...
    repeat
        ...
        send eingaben to auftragnehmer;
        receive ausgaben from auftragnehmer;
        ...
    end repeat
end process

auftragnehmer: process
    eingabe : ...
    ausgabe : ...
    auftrag(...);
```



```
        -- Auftrag bearbeiten
    repeat
        ...
        receive eingaben from auftraggeber;
        auftrag(eingaben, ausgaben);
        send ausgaben to auftraggeber;
        ...
    end repeat
end process
```

- RPC

```
auftraggeber: process
    eingaben : ...
    ausgaben : ...
    repeat
        ...
        auftragnehmer.auftrag(eingaben, ausgaben);
        ...
    end repeat
end process

auftragnehmer: process
    export auftrag;
    auftrag(...);
        -- Auftrag bearbeiten
end process
```

✎ Eigenschaften

1. einfache Bedeutung
2. strukturierte Programmierung
3. effiziente Implementierung
4. allgemeine Anwendbarkeit

✎ Nebenläufigkeit bei Prozedurfernaufrufen

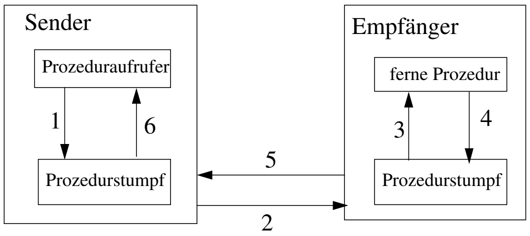
➪ Druckaufträge und Drucker

✎ Implementierung von Prozedurfernaufrufen

Für die Umsetzung des Prozeduraufrufs in einen Nachrichtenaustausch sorgt ein lokaler Prozedurstumpf (*procedure stub*), welcher die ferne Prozedur im Adreßraum des Aufrufers repräsentiert.

• Aufgaben des Prozedurstumpfs auf der Senderseite
1. Zusammenstellen von Nachrichten zum Beantragen einer fernen Prozedur
2. Senden von Nachrichten
3. Warten auf Rückmeldungen
4. Weiterleiten von Ausgabeparametern an den Aufrufer

• Aufgaben des Prozedurstumpfs auf der Empfängerseite
1. Entgegennehmen von Nachrichten
2. Aufrufen der beantragten Prozedur
3. Zurücksenden von Rückgaben als Nachrichten an den Aufrufer



• Parameterübergabe

1. call by value: der Wert wird als Nachricht an den Auftragnehmerprozeß geschickt
2. call by name: der Wert der Variablen wird gesendet
3. call by value result: der Wert der Variablen wird gesendet, und nach dem Abarbeiten der fernen Prozedur wird ein veränderter Wert zurückgeschickt und der Variablen zugewiesen
4. call by reference: problematisch

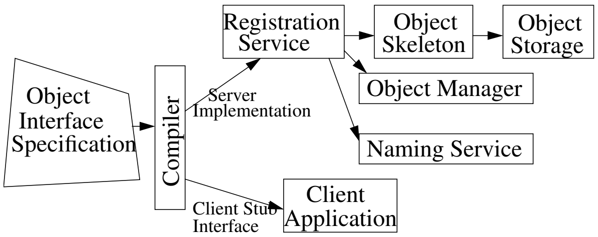
➪ rpc-Bibliothek für C-Programme

#include <rpc/rpc.h>)

4.2 Verteilte Objekte

In einem verteilten Objektsystem (*distributed object system*) sind die Objekte über ein Netz verteilt. Es ermöglicht die direkte Interaktion zwischen den Objekten.

✎ Allgemeine Architektur verteilter Objektsysteme



• Object Interface Specification: bietet die Unabhängigkeit zwischen der Benutzung von Objekten und deren Implementierung.

➪ IDL - Inferface Definition Language in CORBA
COM - Component Object Model in DCOM

• Compiler: erzeugt die implementierungsorieniterten Inferfaces.
• Object Manager: verwaltet die Objekt-Skeletons und Objekt-Referenzen.

➪ ORB - Object Request Broker in CORBA

➪ java.rmi.registry.* in Java

• Registration Service: meldet die Implementierung eines Interface an, so daß die Klienten sie adressieren können,

• Naming Service: weist die Aufträge von Klienten an das richtige Objekt.

• Object-Skeleton

Ein Object-Skeleton ist eine Schnittstelle zwischen der Objektimplementierung und dem Objektmanager. Es wird auf der Serverseite verwendet für die
- Erzeugung neuer Objekte der Klasse, und
- Transformieren eines Methodefernaufrufs zu dem passenden Objekt.

• Client-Stub

Ein Client-Stub ist eine Schnittstelle zwischen der Objektimplementierung und dem Client. Es wird von Klienten verwendet um Methodeaufrufe an das Objekt auf der Serverseite zu lenken.

✎ Eigenschaften verteilter Objektsysteme

• wiederverwendbare Objekte
• erweiterbare Systeme
• einfache Implementierung komplexer Anwendungen
• leichte Instandhaltung