Middleware für verteilte industrielle Umgebungen

Remote Procedure Call

(Client-Server-Modell)

ifak

IPC – Interprocess Communication

- Client/Server- und Gruppen(mehrteilnehmer)kommunikation
- eine "Bedienungsanleitung“ für Netzwerkprotokolle
  - high-level Protokolle zur Prozessinteraktion
  - Grundlage bilden Transportprotokolle, z.B. TCP und UDP
  - kommunizierte Informationen stellen Datenströme dar
    - die keine Nachrichtengrenzen aufweisen
    - deren Transfer (i.A.) gepuffert erfolgt
  - Abstraktionen zum message passing
- die zwei zentralen Fragestellungen beziehen sich dabei auf . . .
  - 1. die Repräsentation zu übertragender Datenstrukturen
    - d.h. der Datentypen der Anwenderprogramme
  - 2. die Kommunikation zwischen entfernten Prozessen
    - problemorientierte, anwendungsbezogene Protokolle
- . . . End-to-End Arguments in System Design
  - Vermeidung redundanter Nachrichten und Funktionen

Abbildung der Datenelemente

- die Datenelemente bilden sich aus Datenstrukturen
  - d.h. sie stellen strukturierte Daten dar
  - dabei handelt es sich um einfache, elementare Datentypen . . .
    - char, int, unsigned, double
  - . . . aber auch um komplexe, dynamische Datenstrukturen
    - struct, union, class
    - Listen, Bäume, Graphen
- Nachrichten sind nichts weiter als linearisierte Daten
  - Datenstrukturen sind in eine „flache Form“ zu bringen . . .
    - sie werden vor dem Versand in einen Puffer kopiert
  - . . . und in ihren Originalzustand wieder hergestellt
  - sie werden nach dem Empfang aus einen Puffer kopiert
- eine externe Datenrepräsentation ist erforderlich
  - Nachrichten enthalten Daten beliebiger Typen
  - Typen sind nicht überall gleich repräsentiert

Datenrepräsentation

- Möglichkeiten, ein einheitliches Verständnis zu erlangen:
  1. Konvertierung der Daten in eine externe Darstellung
     - Sender wandelt lokale Daten um in die externe Form
     - Empfänger wandelt externe Daten um in die lokale Form
  2. verbindungsorientierte Kommunikation und Aushandlung
     - eine Umwandlung ist nur bedingt erforderlich . . .
       - wenn z.B. die Prozessoren inkompatibel zueinander sind
     - . . . ihre Notwendigkeit wird ausgehandelt
     - beim Verbindungsaufbau
  3. Attributierung der Nachrichten mit dem Prozessortyp
     - der Empfänger wandelt empfangene Daten um . . .
     - . . . wenn seine CPU zum angegebenen Prozessortyp passt
- in allen Fällen liegt eine externe Datenrepräsentation vor

Externe Datenrepräsentation

- zwei Kategorien werden unterschieden . . .
  1. getypte Datendarstellung in den Nachrichten
     - ASN.1 (abstract syntax notation) (CCITT-Standard)
     - der Typ der Datenelemente ist genau spezifiziert
  2. ungetypte Datendarstellung in den Nachrichten
     - Sun XDR (external data representation), Xerox Courier
       - Firmenstandards
     - der Typ der Datenelemente ist nicht spezifiziert
       - spezifiziert wird die Struktur der Elemente
     - die Daten werden als Bytestrom übermittelt
- . . . die beide zu Lasten der Bandbreite gehen
  - wegen der zusätzlich zu transferierenden Informationen

ifak

XDR Nachrichten

- Nachrichten stellen "Quartett“-Sequenzen dar
  - Objekte fester Länge (4 Bytes)
  - folgende Konventionen werden eingeführt:
    1. Zahlen belegen ein Objekt (32 Bits)
    2. Zeichenfolgen von 4 Bytes Länge belegen ein Objekt
    3. Felder, Strukturen, Zeichenketten sind eine Bytefolge
       - mit einem anführenden Längenfeld ("offene Felder“)
    4. Zeichen liegen im ASCII vor

5	Sequenzlänge
„Ried“	Riedl
„l "	
7	Sequenzlänge
„Wahl“	Wahlitz
„itz“	
2009	unsigned

ifak

XDR Nachrichten (ff)

- weitere Konventionen legen fest:
  5. das höchstwertigste Ende eines Objektes
  6. ob Zeichen gepackt vorliegen
  7. welches von den 4 Bytes zuerst übertragen wird

❐ die feste Objektgröße senkt Verarbeitungskosten

- allerdings auf Kosten der Übertragungsbandbreite

ifak

Marshalling

- Eine "Datenkollektion“ zu einer Nachricht verpacken
  - umfasst zwei grundlegende Schritte . . .
    1. Linearisierung der Datenstrukturen
       - einfach bei statischen Datenstrukturen
       - aufwendig bei dynamischen Datenstrukturen
    2. Umwandlung in die externe Repräsentation
  - . . . die manuell oder automatisch vorgenommen werden
    - von Hand z.B. mit `sprintf()`:

```
const MAX = 1024;
char buf[MAX];
char* name = "Riedl", ort = "Wahlitz";
sprintf(buf, "%u%s%u%s%u",
        strlen(name), name,
        strlen(ort), ort,
        2009);
```

ifak

Marshalling (ff)

    - automatisch i.A. nur im Zusammenhang mit Fernaufrufen
  - der umgekehrte Vorgang bezeichnet sich mit "unmarshalling“
    - findet auf der Empfängerseite statt
- am Beispiel von XDR:
  1. Spezifikation des Nachrichtenformates

```
const MAX = 8;
struct String {
    unsigned size;
    char data[MAX];
};
struct Message {
    String name;
    String place;
    unsigned year;
};
```

Marshalling (ff)

2. Zusammenstellen der Nachricht

```
char* name = "Riedl", ort = "Wahlitz";
Message buf;
buf.name.size = strlen(name);
strcpy(buf.name.data, name);
buf.place.size = strlen(ort);
strcpy(buf.place.data, ort);
buf.year = 2009;
```

- der Programmieraufwand ist noch recht erheblich

ifak

Kommunikationsprimitiven

- `send()` und `receive()` von (beliebigen) Nachrichten
  - in zwei möglichen Ausführungen . . .
    1. synchrone Kommunikation
       - Datentransfer erfolgt abhängig von Prozesszuständen
       - Sende- und Empfangsprozess blockieren dazu ggf.
       - Ende-zu-Ende Semantik
    2. asynchrone Kommunikation
       - Datentransfer erfolgt unabhängig von Prozesszuständen
       - Sende- und Empfangsprozess blockieren (i.d.R.) nicht
       - keine Ende-zu-Ende Semantik
  - . . . die Nachrichtenwarteschlangen bedingen
    - aber nicht unbedingt Nachrichtenpuffer

Kommunikationsprimitiven (ff)

- gepufferte Kommunikation impliziert blockierende Operationen
  - wenn Nachrichtenpuffer voll oder leer sind
  - wenn keine wiederverwendbaren Betriebsmittel frei sind . . .
  - . . . um konsumierbare Betriebsmittel zu verwalten
  - Eine "Auszeit“ (timeout) kann die Blockade aufheben
- "synchron“ muss nicht immer "blockieren“ bedeuten

ifak

Kommunikationsadressen

- repräsentieren den Bestimmungsort von Nachrichten
  - müssen im `send()/receive()` spezifiziert werden
- der Bezeichner (identifier) des Bestimmungsortes . . .
  - muss den Sendeprozessen bekannt sein
  - kann den Empfangsprozessen bekannt sein
- . . . identifiziert den Empfangsprozess direkt oder indirekt

ifak

Kommunikationsadressen (ff)

- drei Varianten sind gebräuchlich:
  1. direkte Kommunikation
     - von Prozess zu Prozess
  2. indirekte Kommunikation
     - von Prozess über Port zu Prozess
  3. verbindungsorientierte Kommunikation
     - von Prozess über Verbindung zu Prozess
     - die Verbindung besteht zwischen Ports
- trade-off zwischen Performanz und Flexibilität/Transparenz

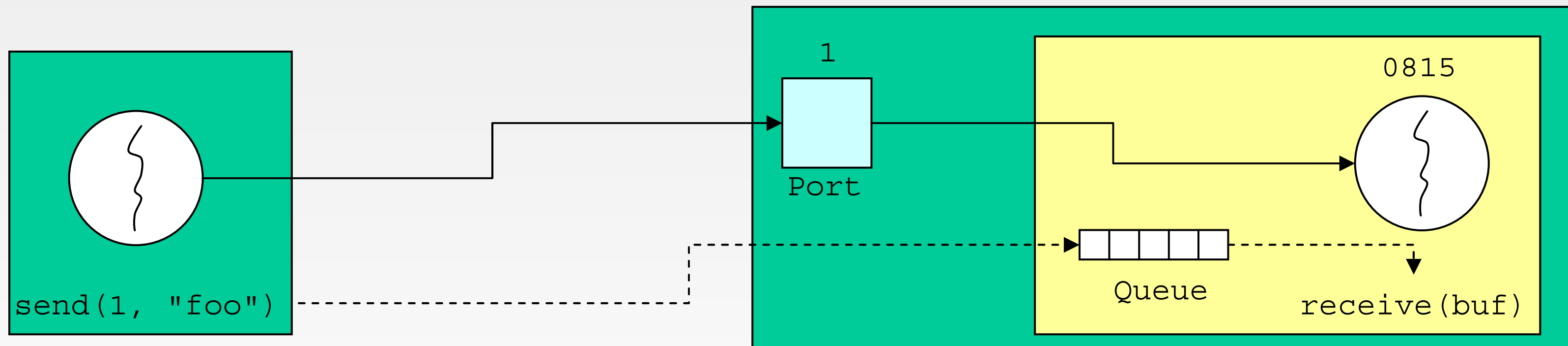
ifak

Ortsunabhängige Bezeichner

- repräsentieren (Ziel-) Adressen für zu sendende Nachrichten
  - formaler Parameter der `send()`-Primitive
- sind abzubilden auf ortsabhängige Bezeichner . . .
  - Aufgabe der Software zur Wegewahl (routing)
- . . . um Nachrichten ihrem Bestimmungsort zuzuführen
  - einem Prozess oder Briefkasten (mailbox) zuzustellen
  - ggf. durch ein Tor (port) weiterzuleiten
- die bezeichneten Objekte werden dadurch "umsetzbar“
  - Server können z.B. ihre Lokalität verändern . . .
  - . . . ohne Klienten den neuen Ort mitteilen zu müssen
- Prozessmigration unter Beibehaltung der Ortstransparenz

Port versus Mailbox

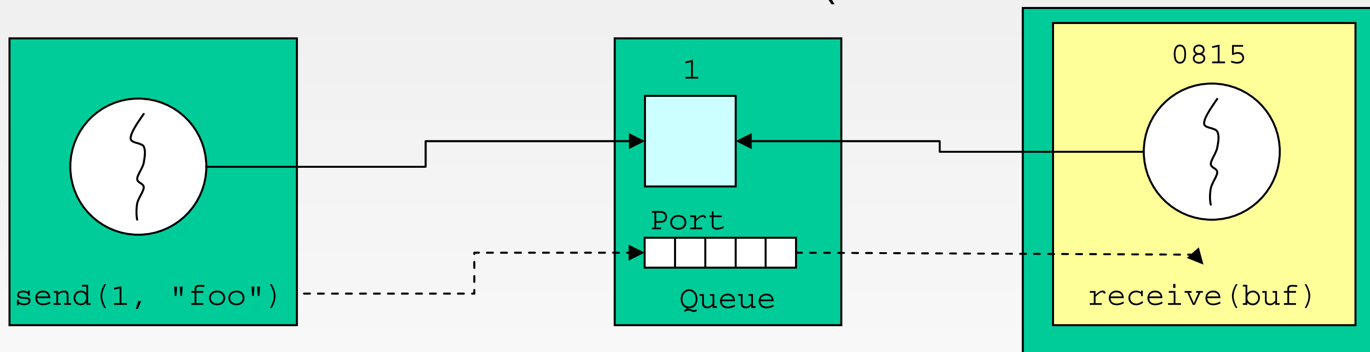
- der Port dient der Nachrichtenweiterleitung
  - mit ihm ist keine Nachrichtenwarteschlange verbunden



- die Nachrichten werden "unverzüglich“ dem Prozess zugestellt
  - der Prozesskontrollblock kapselt die Nachrichtenwarteschlange
- der Port verweist auf den ihm angebundenen Prozess
  - durch eine Adresse auf den Prozesskontrollblock . . .
  - . . . oder den (systemweit eindeutigen) Prozessbezeichner

Port versus Mailbox (ff)

- die Mailbox dient der Nachrichtenaufbewahrung
  - sie ist ein Port mit einer Queue (Nachrichtenwarteschlange)



- Prozesse müssen Nachrichten aus ihren Briefkästen abholen
  - dabei ist die Mailbox explizit zu spezifizieren

ifak

Nachrichtenbezeichner

- zur Erhöhung der Zuverlässigkeit der Kommunikation
  - zuverlässige Nachrichtenzustellung (reliable message delivery)
  - request-reply Kommunikation zwischen Client und Server
- die eindeutige Identifikation der Nachrichten ist zweiteilig:
  1. Kennung der Anforderung (request id)
     - bei jedem send() um 1 erhöhter Zählerwert
     - wird vom Sendeprozess generiert
  2. Kennung des Sendeprozesses (sender id)
     - Prozess- oder Portbezeichner
       - um eventuelle Rücknachrichten empfangen zu können
     - Adresse (d.h. Parameter) der reply() Primitive

ifak

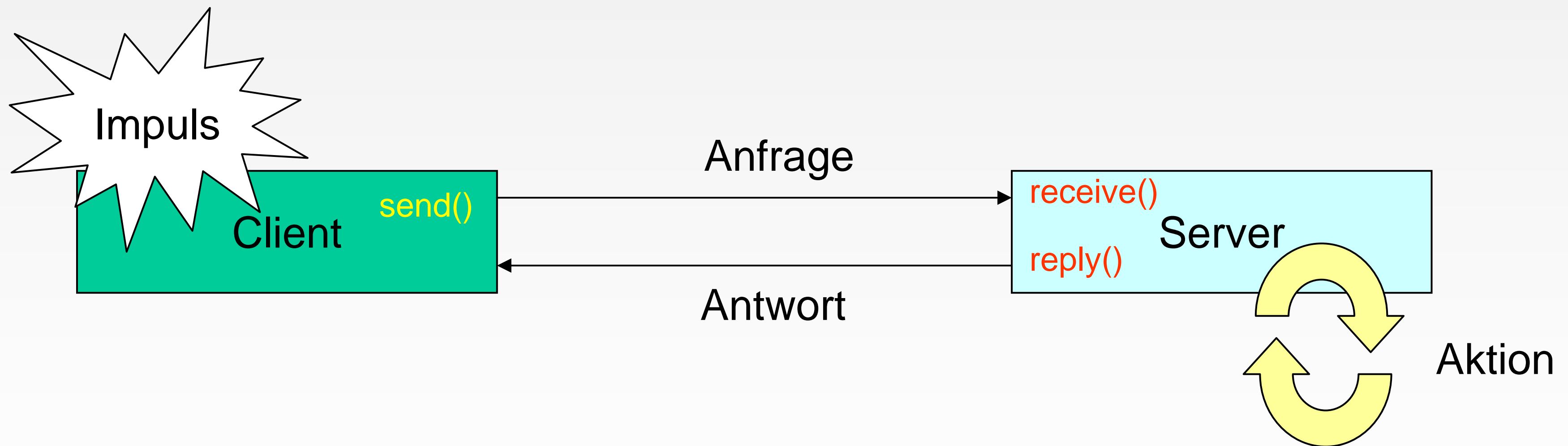
Nachrichtenbezeichner (ff)

- die request_id macht die Nachricht lokal eindeutig
  - allerdings nur für den jeweiligen Sendeprozess
- die sender_id macht die Nachricht global eindeutig
  - speziell für den jeweiligen Empfangsprozess
  - generell für das verteilte System

ifak

Client-Server-Kommunikation

- „Normalform“ ist synchrone Kommunikation
  - der Client fordert, blockiert und wartet auf eine Antwort
  - request-reply



Client-Server-Kommunikation (ff)

- request-reply Protokolle basieren auf drei Primitiven:
  1. `send()` (bzw. DoOperation())
     - nicht-blockierender Versand der request-Nachricht
     - blockierender Empfang der reply-Nachricht
  2. `receive()` (bzw. GetRequest())
     - blockierender Empfang der request-Nachricht
  3. `reply()` (bzw. SendReply())
     - nicht-blockierender Versand der reply-Nachricht

Aufbau einer request-reply-Nachricht

- unterscheidet typischerweise vier Einträge:
  1. Nachrichtentyp
     - request oder reply
  2. Nachrichtenbezeichner
     - sendeseitig generierte request id
  3. Operationskode
     - Kennung der empfangsseitig auszuführenden Funktion
  4. Parameter
     - linearisierte Datenstrukturen (aktuelle Ein-/Ausgabewerte)
- beschreibt einen Fernaufruf (*remote procedure call*, RPC)

ifak

Protokollvarianten

1. request (R) Protokoll
   - wird eingesetzt, wenn kein "Rückgabewert“ erwartet wird
   - der Client setzt seine Ausführung nach dem Versand fort
2. request-reply (RR) Protokoll
   - spezielle Bestätigungsnachrichten werden nicht benötigt
     - das reply() bestätigt implizit die request Nachricht
     - ein weiteres send() bestätigt die reply Nachricht
   - nur zwei Nachrichten werden ausgetauscht
3. request-reply-acknowledge reply (RRA) Protokoll
   - die reply Nachricht wird explizit bestätigt
     - die request id der reply Nachricht wird zurückgeschickt
     - Blockbestätigung für alle "niederwertigen“ replies
   - drei Nachrichten werden ausgetauscht

Mögliche Fehler

- duplizierte *request* Nachrichten erkennen und ignorieren
  - duplicate suppression protocol
  - Problem sind die möglichen "Auszeiten“ (timeouts) . . .
  - . . . d.h. die explizit wiederholten request Nachrichten
- verlorene *reply* Nachrichten behandeln
  - eine ggf. bereits gesendete Antwort nochmals anfordern
    - den *request* dabei nur bedingt noch einmal bearbeiten
  - idempotente Operationen bzw. Zustandsfreiheit . . .
    - . . . Buch über bereits geschickte *reply* Nachrichten führen
- alles Probleme mit Ende-zu-Ende-Relevanz

ifak

RPC – Remote Procedure Call

- Erweiterung des Prozeduraufrufs zum Fernaufruf
- Ziel: Syntaktische und semantische Uniformität
  - Aufrufmechanismus
  - Sprachumfang
  - Fehlerfälle
- Definition (nach Nelson)
  - Synchrone Übergabe des Kontrollflusses
  - Ebene der Programmiersprache
  - Getrennte Adressräume
  - Kopplung über relativ schmalen Kanal
  - Datenaustausch: Aufrufparameter und Ergebnisse

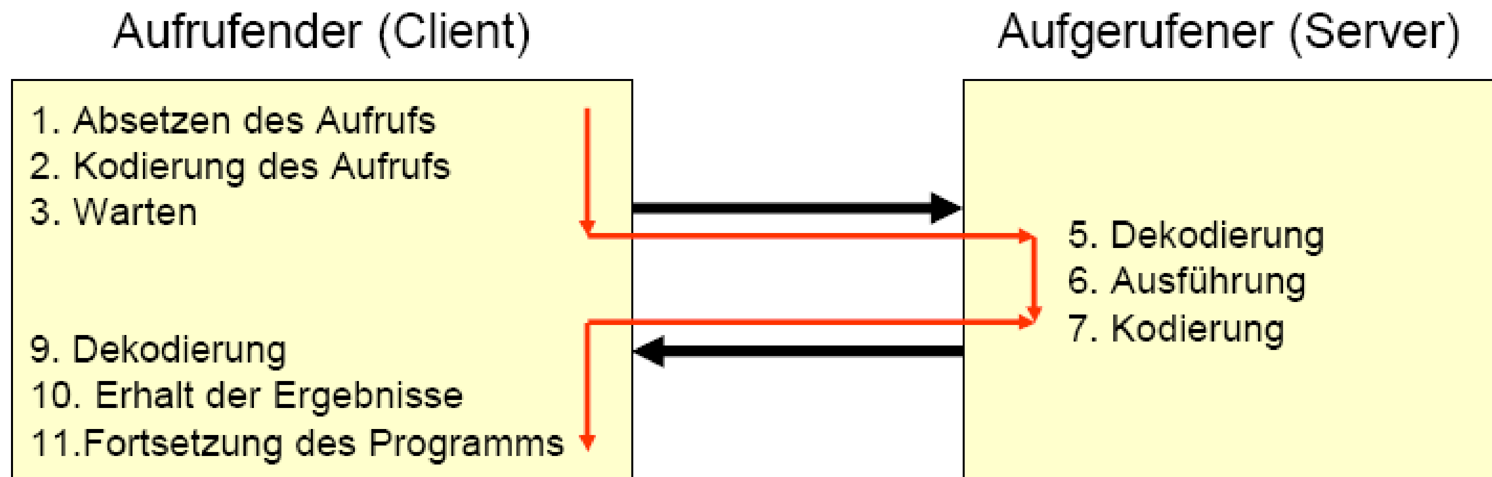
Eigenschaften

- Semantik konventioneller Prozeduraufrufe aufrechterhalten . . .
  - . . . obgleich die Ausführungsumgebung radikal anders ist
- die Semantik lokaler Aufrufe ist nur teilweise erreichbar
  - aufrufende und aufgerufene Ebene sind örtlich getrennt
    - sie besitzen keinen gemeinsamen Adressraum
    - Code und Daten liegen nicht im selben Arbeitsspeicher
  - beide Ebenen arbeiten weitestgehend autonom
    - sie sind verschiedenen Rechnern zugeordnet
    - sie werden von verschiedenen Prozessen ausgeführt
  - beide Ebenen können unabhängig voneinander ausfallen
    - der Client, während der Server seinen Aufruf bearbeitet
    - der Server, während der Client eine Antwort erwartet
  - die Sicht "alles-oder-nichts“ ist nicht gegeben
- spezielle Fernaufrufprotokolle sind erforderlich

ifak

RPC – Remote Procedure Call

- Ablauf
  - Aufrufer im Wartezustand
  - Parameter- und Aufrufübertragung ins Zielsystem
  - Prozedurausführung
  - Rückmeldung
  - Fortsetzung der Programmausführung



ifak

Semantikaspekte

- Parameter
  - Eingabeparameter
    - client-seitig Werte in die request-Nachricht eintragen
    - server-seitig Werte der request-Nachricht entnehmen
  - Ausgabeparameter
    - server-seitig Werte in die reply-Nachricht eintragen
    - client-seitig Werte der reply-Nachricht entnehmen
  - Ein-/Ausgabeparameter
    - in *request*- und *reply*-Nachricht transportieren
- Parameterübergabe
  - call-by-value ist "trivial“
  - call-by-reference erfordert zusätzliche Maßnahmen
    - "Eingabe“ und/oder "Ausgabe“ ist explizit zu spezifizieren

ifak

Semantikaspekte (ff)

- interface definition language (IDL)
- Gültigkeitsbereiche
  - Variablen der aufrufenden Umgebung sind ungültig
  - örtlich voneinander getrennte Ausführungsumgebungen
- Speicheradressen
  - sind systemweit uneindeutig
  - Zeiger in Nachrichten zu übertragen, ist (nahezu) sinnlos

ifak

Systemarchitektur

- RPC-DEC (Distributed Computing Environment)

Netz

Client-Rechner

Server-Rechner

Client | Client-Stub | Laufzeit-system

lokaler Aufruf → Aufruf-kodierung → sende → warten

lokales Ergebnis ← De-kodierung ← empf.

Ruf-Paket →

← Ergebnis-Paket

Laufzeit-system | Server-Stub | Server

empf. → De-kodierung → Aufruf → Aus-führung

sende ← Kodierung ← Ergebnis

Import ← ... ← **Export**

ifak

Kommunikation

Directory Service

Import

Export

2

1

Bindevorgang

Client
(z.B. Kasse)

Server
(z.B. Kontenserver)

Objekt-
interaktion

Schnittstellen-
spezifikation

Prüfziffernrechnung

Prüfziffernrechnung

Info-Datei

Info-Datei

3

Status

4

Status

5

10010111...

Basiskom-
munikation

Schnittstellenbeschreibung: IDL

```
[uuid(765c3b10-100a-135d-1568-040034e67831),version(1.0)]

interface DocumentServer      // Schnittstelle für Dokumenten-Server
{
  import "globaldef.idl";        // Import allg. Definitionen
  const long maxDoc=10;          // Maximale Produktanzahl
  typedef [string] char *String; // Datentyp für Character-Strings
  typedef struct {
    String documentName;         // Dokumentname
    String documentDescription;  // Textuelle Beschreibung
    long size;                   // Speicherumfang
  } DocumentDescription;         // Dokumentbeschreibung
  typedef struct {
    DocumentDescription desc;    // Dokumentbeschreibung
    String header;               // Dokumentkopf
    char *data;                  // Dokumentdaten
  } Document;                    // Dokument
```

ifak

Schnittstellenbeschreibung: IDL

```
[idempotent] long documentQuery ( // Anfrage nach Dokumenten
  [in] String documentName[maxDoc], // Dokumentnamen
  [out] DocumentDescription *dd[maxDoc], // Beschreibungen
  [out] long *status);             // Statuswert der Operation

long insertDocument (              // Dokument einfügen
  [in] Document *d,                // Neues Dokument
  [out] long *status);             // Statuswert der Operation

long removeDocument (              // Dokument löschen
  [in] String name,                // Dokumentname
  [out] long *status);             // Statuswert der Operation

long fetchDocument (               // Beschaffen eines Dokuments
  [in] DocumentDescription *dd,    // Dokumentbeschreibung
  [out] Document *d);              // Gesuchtes Dokument
}
```

ifak

Stub-Generierung

document.idl

IDL-Compiler

document_cstub.o

document.h

document_sstub.o

#include

document_client.c

document_server.C

C compile

document_client.o

document_server.o

link

link

document_client.exe

document_server.exe

ifak

Bindevorgang

Alternativen :
- Direktadressierung
- Broadcast-Anfrage
- Directory Dienst

Directory-Server

Verwaltungs-tabelle

Client

Import DocumentServer

Serveradresse S

Binde (S, DocumentServer)

RPC:
fetchDocument(desc,document)

Server

Export DocumentServer
- documentQuery
- insertDocument
- removeDocument
- fetchDocument

Quittiere(Binde-Nummer)

return(document);

Explizites Binden durch Client

Client

rpc_ns_binding_import_begin

Kontext

rpc_ns_binding_import_next

Binding Handle

rpc_ns_binding_import_done

Directory Service

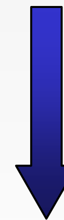
Namenseintrag :

/.:/DocumentServer

ifak

Details beim Binden

- Caching von Binde-Information
  - Information auf Client-Seite global für alle Prozesse
  - Erkennung veralteter Information (z.B. Timeout)
  - Begrenzte Informationshaltung auf Serverseite (Skalierbarkeit, Wiederanlauf)
- Zeitpunkt des Bindens
  - Übersetzungszeitpunkt
  - Linkzeitpunkt
  - Dynamisch
  - Gemischte Verfahren
    - Logische Namen
    - Erste Lokalisierung beim Binden zur Initialisierungszeit
    - Erneutes Lokalisieren bei Fehlern



Flexibilität

Aufwand

ifak

Transportprotokolle bei DCE

- TCP / IP
- UDP / IP
- Herstellerspezifisch
- Beispiel:

```
rpc_server_use_protseq(„ncacn_ip_tcp“,
                       rpc_c_protseq_max_reqs_default,
                       &status);
rpc_server_use_all_protseq(rpc_c_protseq_max_reqs_default,
                           &status);
```

ifak

Laufzeitunterstützung: Prozesse

- Prozessverwaltung
  - Zuordnung von Prozessen zu Prozedurausführungen, Lösung der Verklemmungsproblematik
  - „Lightweight“ Prozesse (Threads) :
    - Gemeinsamer Adressraum
    - Schnelle Erzeugung und Prozessumschaltung
    - Große Anzahl von Prozessen möglich

    → Einsatz bei RPC - Server - Implementierungen

    → Einsatz auch bei Client → asynchron
  - Prozesszuordnung
    - Prozesserzeugung pro Aufruf oder
    - Prozess - Pool
  - Pufferweitergabe über Referenzen über weitere Protokollschichten

    → Effizienz

ifak

Prozesse auf Client-Seite

- Prozesserzeugung :
  - explizit durch `pthread_create()`
  - Übergabe von Startroutine und Parametern
  - Felder von Prozessen möglich
- RPC :
  - separat in Threads eingebettet
  - Rückgabewert über `pthread_exit()`
- Synchronisation
  - blockierendes `pthread_join()`
  - separat für alle Threads

ifak

Fehlerbehandlung beim RPC

- Fehlerfälle :
  1. Fehler während der Prozedurabarbeitung
  2. Übertragungsfehler
- Fehlerursachen
  - Ausfall eines beteiligten Rechners
    - Server-Seite → endloses Warten des Clients → Timeout
    - Client-Seite → Weitere Bearbeitung als „Orphan“
  - Unerreichbarkeit des Zielknotens

    → dynamisches Binden
  - Veraltete Prozedur - Export - Information

    → dynamisches Binden

ifak

Fehlerbehandlung beim RPC (ff)

- Fehlersemantik (Spector) :
  - **Maybe**
    - Höchstens einmalige Ausführung ohne Benachrichtigung
    - im Fehlerfall → nur für „unwichtige“ Operationen
  - **At - least - once**
    - Mindestens einmalige Ausführung
    - nur bei idempotenten Operationen
  - **At - most - once**
    - Erkennung und Löschen von Duplikaten
    - Ausführung nur, wenn keine Rechnerausfälle vorliegen
  - **Exactly - once**
    - Ausführung genau einmal
    - auch bei Rechnerausfällen
    - → Transaktionskonzepte mit Wiederanlauf von Komponenten

ifak

Fehlersemantik

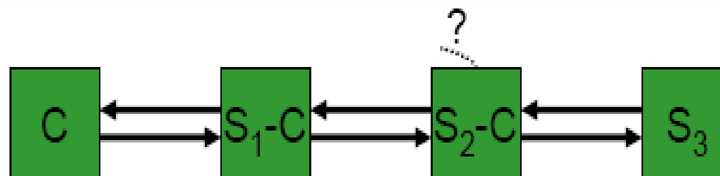
Fehlerklassen \ Fehlerarten	Fehlerfreier Ablauf	Nachrichten-verluste	Zusätzlich Ausfall des Servers	Zusätzlich Ausfall des Clients
Maybe	Ausführg.: 1 Ergebnis: 1	Ausführg.: 0/1 Ergebnis: 0/1	Ausführg.: 0/1 Ergebnis: 0/1	Ausführg.: 0/1 Ergebnis: 0/1
At-Least-Once	Ausführg.: 1 Ergebnis: 1	Ausführg.: >=1 Ergebnis: >=1	Ausführg.: >=0 Ergebnis: >=0	Ausführg.: >=0 Ergebnis: 0
At-Most_Once Only-Once-Type-1	Ausführg.: 1 Ergebnis: 1	Ausführg.: 1 Ergebnis: 1	Ausführg.: 0/1 ‖ Ergebnis: 0/1	Ausführg.: 0/1 Ergebnis: 0
Exactly-Once Only-Once-Type-2	Ausführg.: 1 Ergebnis: 1	Ausführg.: 1 Ergebnis: 1	Ausführg.: 1 Ergebnis: 1	Ausführg.: 1 Ergebnis: 1

Fehlersemantik beim DCE RPC

Attribut	Semantik
default: at-most-once	Max. einmalige Ausführung mit Rückgabewert und ggf. expliziter Fehlermeldung
idempotent	Ggf. mehrmalige Ausführung mit Rückgabewert und ggf. expliziter Fehlermeldung
maybe	Ggf. mehrmalige Ausführung ohne Rückgabewert und ohne Fehlermeldung
broadcast	Ggf. mehrmalige Ausführung bei allen passenden Servern mit Rückgabewert des ersten abgeschlossenen Aufrufs

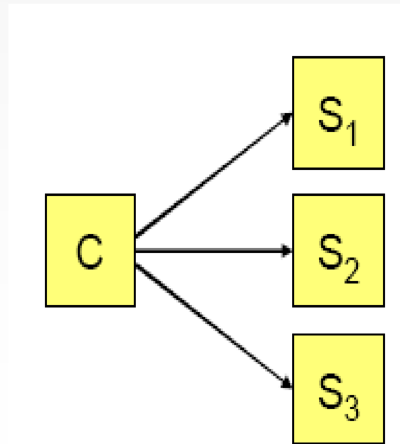
Probleme des RPC

- Übertragung großer Datenmengen
  - Synchroner Mechanismus
  - → kleine Übertragungseinheit
  - Keine verbindungsorientierte Übertragung
  - Keine Flusskontrolle und Pufferung
  - Antwortzeit- statt Durchsatzoptimierung
- Verkettung von Bearbeitungseinheiten
  - Strenge Client- / Server Semantik
  - Keine vorzeitige Datenweitergabe
  - Kein direkter Kontrolltransfer über mehr als 2 Partner



Probleme des RPC (ff)

- Vertauschung von Client- / Server-Rollen
  - Keine gleichberechtigten Kommunikationspartner
  - Keine zwischenzeitlichen Ergebnisrückmeldungen
  - Keine „Rückfragen“
- • Multicast / Broadcast nicht unterstützt



ifak

Probleme des RPC (ff)

- Transparenzverletzungen
  - Variable Parameter und Typenanzahl (z.B. `printf("%s%d",x1,x2);`)
  - Zeigerparameter (z.B. char *x, ...)
  - Globale Variable
  - Fehlersemantik