Praktische Arbeit mit TP-Monitoren

Claus Hagen

Verteilte TP-Monitor-Systeme

Tuxedo (Novell/BEA)

- http://tuxedo.novell.com
- http://www.bea.com

Encina (Transarc)

- http://www.transarc.com

Top End (AT&T GIS / NCR)

- http://www.ncr.com/product/topend

Tuxedo

Ursprünge: Ende der 70er Jahre (AT&T)

Erste kommerzielle Version: 1984

Zunächst rein Mainframe-basiert

Seitdem kontinuierlich weiterentwickelt

Tuxedo: Plattformen

Betriebssysteme

- Server: *ix, OS400, MVS, Windows NT, ...
- Client: zusätzlich Mac, Windows, OS/2

Datenbanksysteme

- Alle "gängigen" DBMS
- (z.B. Informix, Oracle, Sybase, DB/2, ...)

(Voraussetzung: XA-Interface)

Encina

“Encina is reputed to be the Greek Goddess of transaction processing”

Enstand aus einem Forschungsprojekt (Camelot)

Erste kommerzielle Version: 1991

Basierend auf OSF DCE

Encina: Plattformen

Betriebssysteme

- **Server: *ix, Windows NT**
- **Client: zusätzlich Windows**

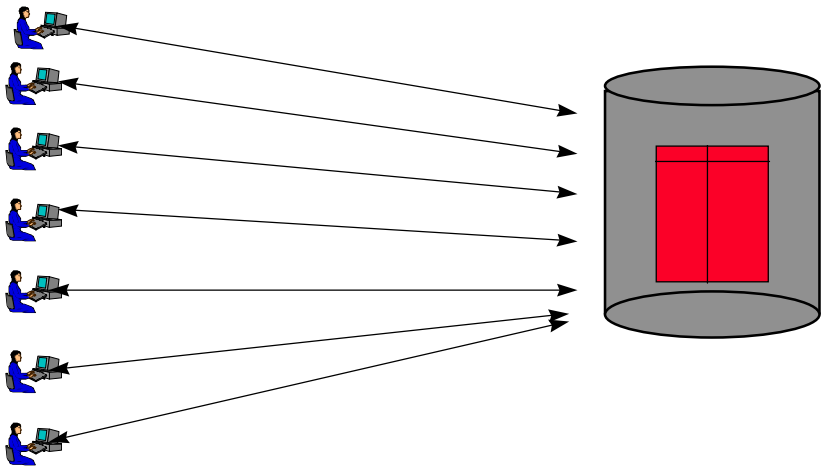
Benötigt OSF DCE

Datenbanksysteme

- **Alle “gängigen” DBMS**
- **(z.B. Informix, Oracle, Sybase, ...)**

(Voraussetzung: XA-Interface)

Beispiel 1



Beispielrelation

ACCOUNTS:

AID	BALANCE
25	735,00
37	942,00
46	10345,00
95	12,50
267	47,92

Typische Operationen:

- Abheben von Geld von Konto 273
- Einzahlen von Geld auf Konto 738
- Ausgabe des Kontostands von Konto 34

Klassischer Ansatz

“Two-Tier” Ansatz

- Direkter Zugriff auf die Datenbank
- Client-Programme enthalten Embedded SQL

Nachteile:

- Keine effektive Zugriffskontrolle
- Keine Kontrolle der Last im RDBMS
- Keine Informationen über Zugriffsarten
- Keine Änderungstransparenz

“Three Tier”-Ansatz

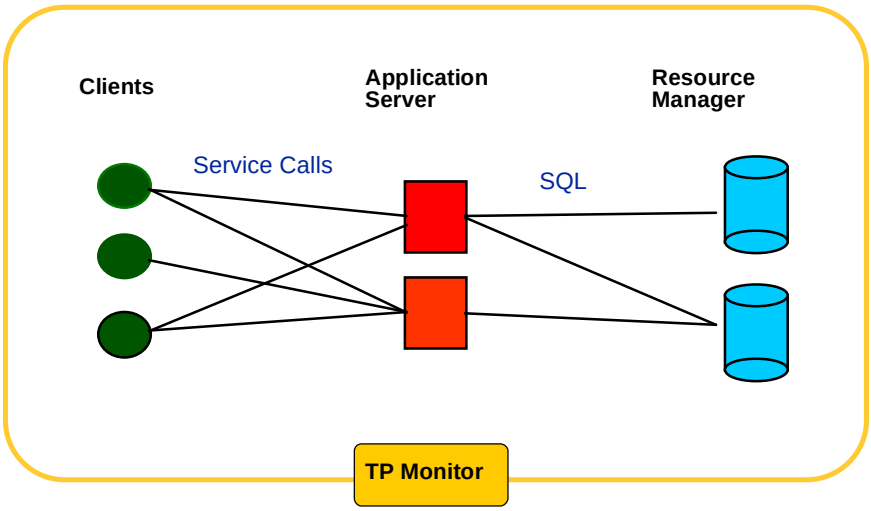
Dreistufige Modellierung des Systems

- Clients
- Anwendungs-Server (Application Server)
- Datenbank (Resource Manager)

Schnittstellen:

- Client - AppServer: Abstrakte Dienste (Services)
- AppServer - Datenbank: SQL

“Three Tier” Ansatz



Resource Manager

Anwendung mit persistentem Zustand

In der Regel Datenbanksysteme

Zugriff ueber standardisierten Befehlssatz (X/Open XA-Interface)

Library mit XA-Interface wird vom DBMS-Hersteller geliefert und muss zum Application Server gelinkt werden.

Application Server

Schnittstelle zwischen Clients und DBMS

“Exportiert” klar definierte Dienste

Beispiel:

- **credit (cid, amount) : real**
- **debit (cid, amount) : real**
- **balance (cid) : real**

Client-Programme

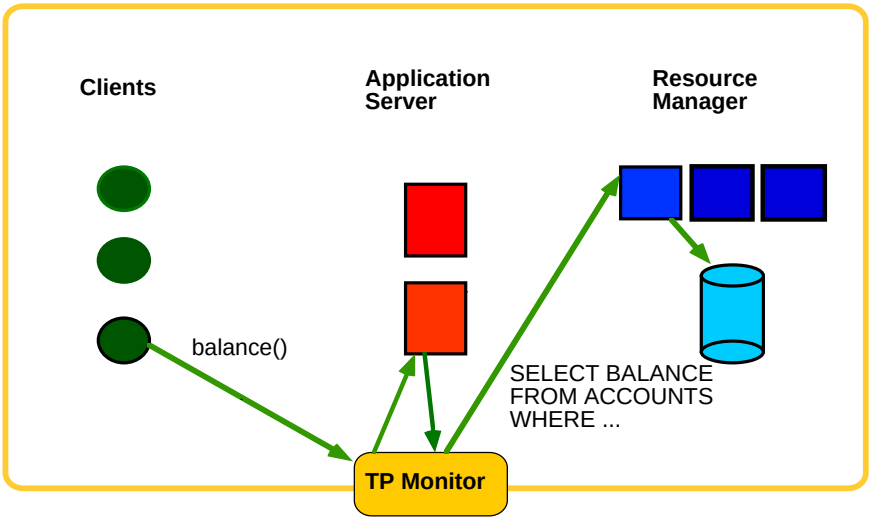
Rufen Dienste der Application-Server auf

“Handgeschriebene” Programme

oder

Automatisch generierte Interfaces (Masken, GUI)

Beispielablauf



Entwicklung eines AppServers

1) Festlegen der Dienste

2) Dienstprogramme erstellen

- C
- C++
- COBOL

3) Kompilierung, Linking

4) Registrierung der Dienste

1) Festlegen der Services

credit (cid, amount) : real

- **Addiert Betrag zum Konto**

debit (cid, amount) : real

- **Zieht Betrag vom Konto ab**

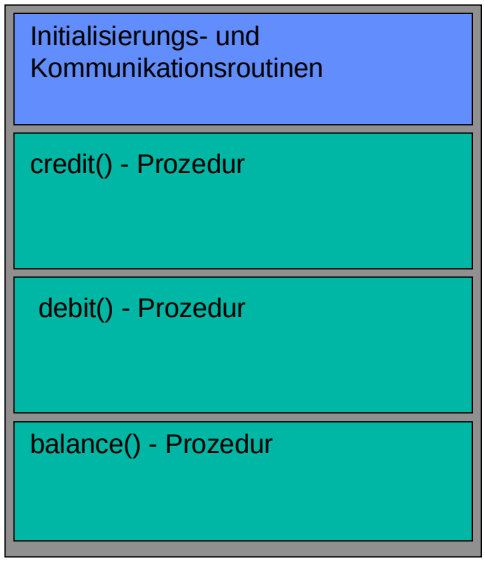
balance (cid) : real

- **Liefert Kontostand**

2) Server-Programm

```
float balance (int account) {
  float bal;

  EXEC SQL
     SELECT BALANCE INTO :bal
     FROM ACCOUNTS
     WHERE AID = :account;

  some_error_handling;
  return bal;
}
```

2) Server-Programm



3-4) Linking, Registration

Linking: Verbinden der Service-Routinen mit speziellen Systemroutinen

Registrierung: Mitteilung an den "Rest der Welt", welche Dienste angeboten werden

Zeitpunkt der Registrierung:

- **Beim Start des Servers**
- **Dynamisch**

Interfaces

Beschreibung von Services

- Service-Namen
- Aufrufparameter
- Rückgabewerte

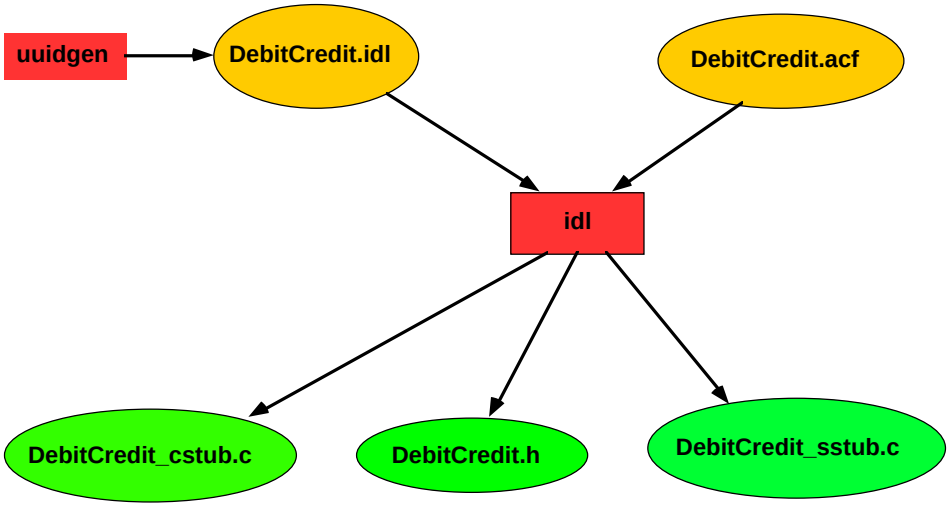
Spezielle Beschreibungssprache

- IDL (Interface Definition Language)

Interfaces sind netzweit eindeutig

- Spezielles Generierungsprogramm

Verarbeitung von IDL-Files



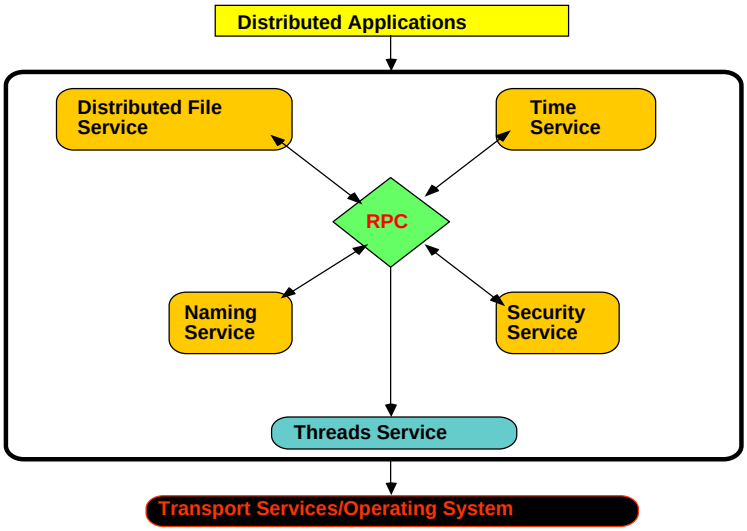
Beispiel: IDL-File

```
[  uuid(002978fe-bb72lea6b3fb9e620404aa77),
   version(1.0) ]

interface CreditDebit
{
   void debit( [in] int account,
               [in] float amount,
               [out] int *newBalP);
   void credit( [in] int account,
               [in] float amount,
               [out] int *newBalP);

   float balance([in] int account);

}
```

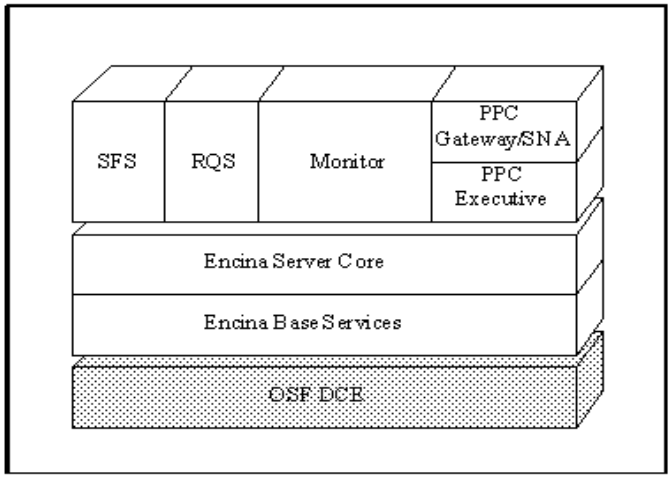
Client-Programm

```
#include <DebitCredit.h>
(...)


float result;

result = balance (acc);
```

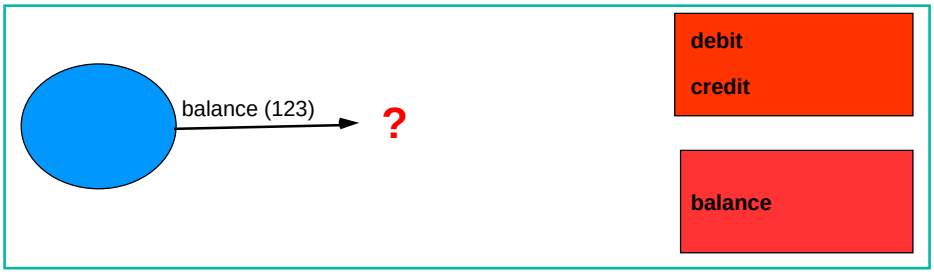
DCE



Encina und DCE



Client-Server-Kommunikation



- **Auswahl des richtigen Servers**
- **Transport von Daten von und zum Server**
- **Datenkonvertierung**
  - Programmiersprachen
  - Architekturen

Client-Programm (Tuxedo)

```
#include <balance.h>
(...)
struct bal_msg *buffer;     /* Puffer deklarieren */
long len;

buffer = (struct bal_msg*) tpalloc ("VIEW", "bal_msg",
0);                  /* Puffer anlegen */

buffer->account = acc;

tpcall ("balance", (char*) buffer, sizeof (struct
MYDATA), (char**) &buffer, &rlen, 0);   /* Aufruf */

result = buffer->bal;

tpfree ((char*) buffer);      /* Puffer freigeben */
```

Buffer (Tuxedo)

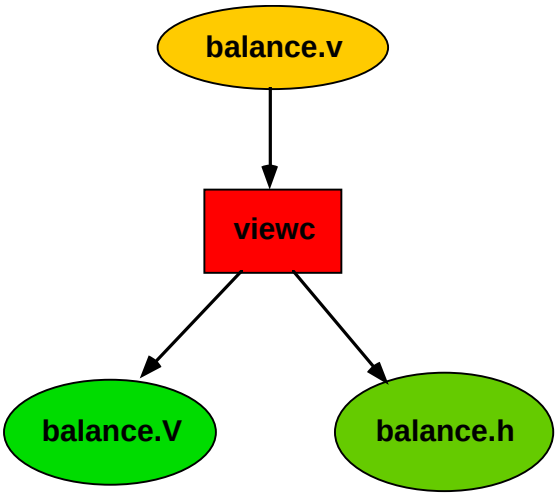
Vordefinierte Datenstrukturen

- String
- CArray
- View
- FML

Automatische Konvertierung

Buffer müssen explizit angelegt werden

Erzeugen von Views

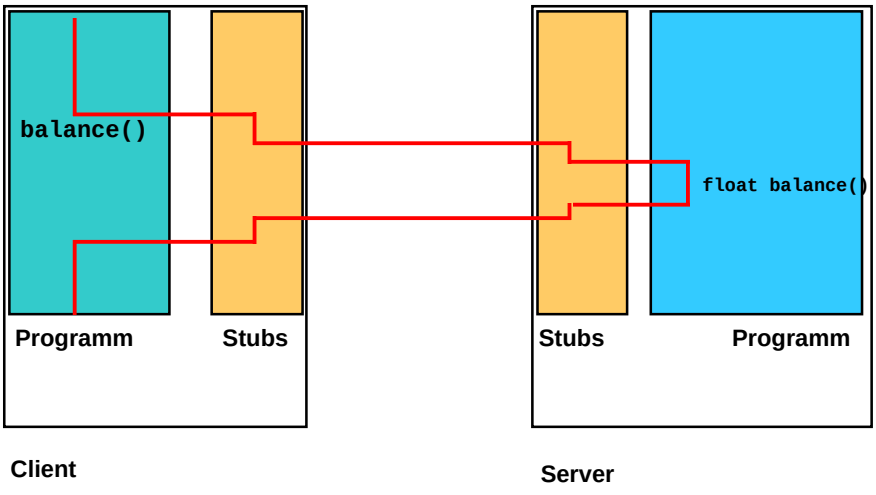


View-Definition

```
$ /* Nachrichtenformat für balance */

VIEW bal_msg

#type    cname     fbname count flag size null

int      account   -       1     -    -    -
float    bal       -       1     -    -    -

END
```

balance.v

RPC (Encina)



RPC (Encina)

Stub-Prozeduren auf Client- und Serverseite

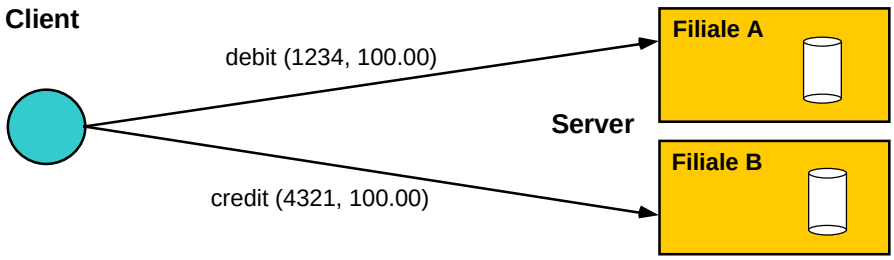
- Konvertierung von Datenstrukturen
- Kommunikation

Stubs werden automatisch erzeugt

- Durch den IDL-Compiler

Einbinden in Programme beim Linken

Verteilte Transaktionen



Korrekte Ausführung verteilter Berechnungen

- Atomarität
- Isolation

Bibliotheksfunktionen (Tux)

tpbegin()

- Start einer globalen Transaktion

tpcommit()

- Erfolgreiches beenden

tpabort()

- Abbruch

Beispiel

```
tpbegin (timeout, flags);

if (tpcall (“debit”, ...) == -1 )
   tpabort(f);
else
   if (tpcall (“credit”, ...) == -1 )
        tpabort(f);
   else
   tpcommit(f);
```

Transactional C, C++ (Encina)

Erweiterung von C bzw. C++

- Macros
- Funktionen
- Klassen

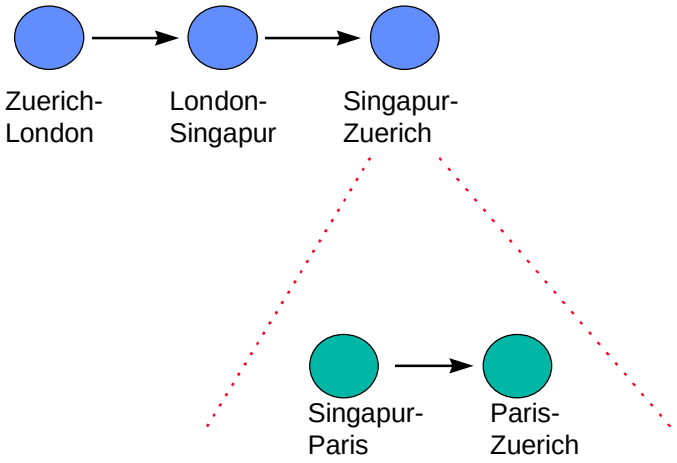
Z.B.:

- Sprachkonstrukte für Transaktionen
- Exception-Mechanismus (C++)

Beispiel

```
transaction {

   debit (1234, 100.00);
   credit (4321, 100.00);
}

onCommit {
        printf ("War erfolgreich \n");
}

onAbort {
        printf ("Oops. Fehler! \n");
}
```

Geschachtelte Transaktionen



Geschachtelte Transaktion

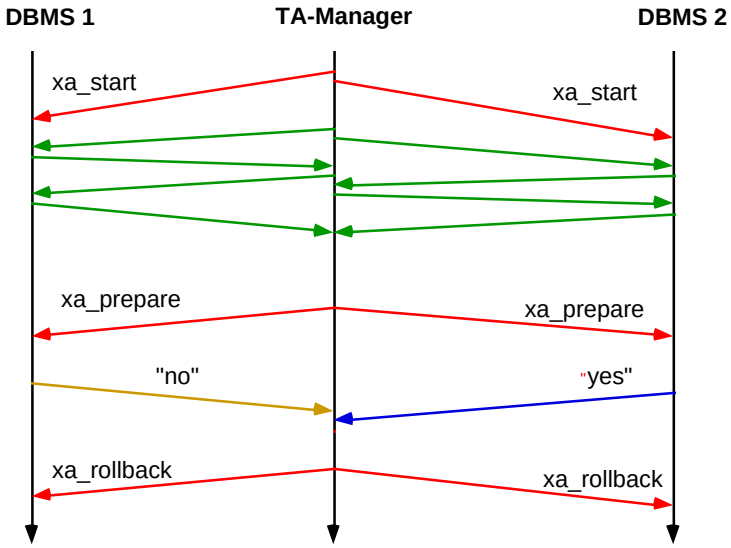
```
transaction {
   Reserviere Zuerich-London
   Reserviere London-Singapur

   transaction {
        Reserviere Singapur-Paris
        Reserviere Paris-Zuerich
   }
   onAbort { * Andere Verbindung ausprobieren * }
}
onCommit {
   /* Bestaetigung drucken */
}
onAbort {* Andere Abflugszeit probieren * }
```

XA-Interface

Standard für die Kommunikation mit DBMS in verteilten Umgebungen

- **Menge von Anweisungen an das DBMS, u.a.:**
  - xa_open(), xa_close()
  - xa_start()
  - xa_prepare()
  - xa_commit()
  - xa_rollback()
- **X/Open Standard**

XA-Interface



Zusammenfassung

Anwendungsentwicklung mit TP-Monitoren

- **Drei-Schichten-Struktur entkoppelt die Anwenderschnittstelle von der Anwendungslogik**
- **Änderungen der Systemkonfiguration beliebig möglich**
- **TP-Monitor uebernimmt Kommunikation und Lastbalancierung**