# Kapitel 7

## *TP-Monitore - Systeme*

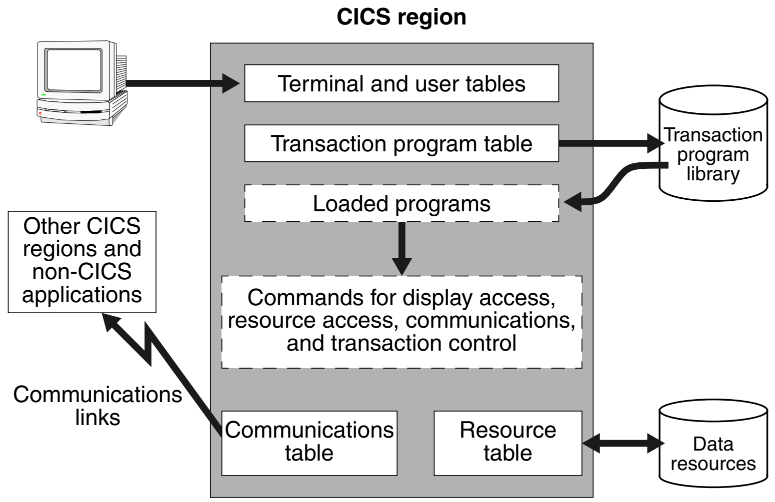
Inhalt

- ❑ CICS
- ❑ TUXEDO
- ❑ ENCINA
- ❑ MTS
- ❑ Zusammenfassung



# CICS (1)

- ❑ Allgemeines
  - ➪ IBM, 1968
  - ➪ größter Marktanteil unter den TP-Monitor-Produkten
  - ➪ urspr. entwickelt für Mainframes
  - ➪ mittlerweile Familie von Produkten auf verschiedenen Code-Basen: CICS/VSE, CICS/MVS, CICS/400 (AS/400), CICS for AIX (auch Digital UNIX, HP-UX, Sun Solaris), CICS for OS/2, CICS for Windows NT
- ❑ Systemarchitektur
  - ➪ Region
    - • Prozess-ähnliche Abstraktion (Adressraum), der mehrere Threads ausführen kann



# CICS (2)

❑ Systemarchitektur (Forts.)

➩ Region (Forts.)

- kontrolliert Applikations-Ressourcen, wie
  - Endgeräte
  - Transaktionsprogramme
  - Datenquellen
  - Kommunikationsverbindungen
- Mainframe CICS
  - TP-Monitor-Threads
  - früher: beschränkte Kommunikationsmöglichkeiten
  - daher meist nur eine Region pro TP-Applikation
- neuere CICS Versionen
  - BS-Threads
  - Empfehlung zur Partitionierung einer Applikation:
    *terminal region* (presentation server),
    *application region* (workflow control),
    *data region* (transaction server);
- Einheit der Fehlerbehandlung (Applikationsfehler)
- Einheit der Verteilung
- Lastbalancierung: *collections of cloned application regions*



# CICS (2)

❑ Systemarchitektur (Forts.)

➩ Abwicklung von Requests

- *Sicherheit*
  - *Terminal table* identifiziert Terminal
  - *geographic entitlement*
  - Benutzerauthentifikation zunächst durch Password, dann mittels *access control list*
  - Benutzerrollen
  - zeitliche Beschränkungen

❑ Systemarchitektur (Forts.)

➩ Abwicklung von Requests (Forts.)

- *Four-Character Transaction Codes*
- *Transaction Routing*: mittels *Transaction Table* wird der Request an zuständige Region weitergeleitet
- (bei Region) eingehende Requests werden
  - nach Priorität sortiert basierend auf Request-Typ, Benutzer und Terminal,
  - Applikationsprogramm wird, falls notwendig, geladen
  - Zuordnung eines Applikations-Threads
- *chained*-Modell (automatischer Start einer Transaktion nach Zuweisung eines Threads)

# CICS (3)

❑ Presentation Server

➪ Möglichkeiten der Kommunikation zwischen Client und Presentation-Server

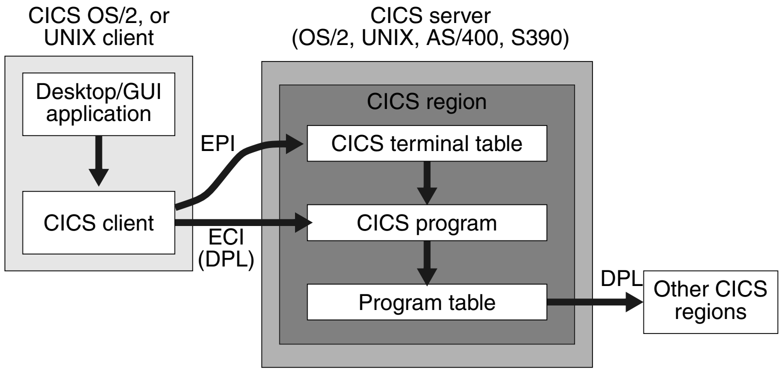
- 3270-Protokoll
  - früher *IBM 3270 Block-Mode Terminals* (Einheit der Übertragung: gesamter Bildschirminhalt)
  - heute noch populär, da viele spezifische Geräte, wie Bankautomaten oder Bar-Code-Leser diese Form der Übertragung noch nutzen
  - evtl. Emulation des *3270 Block-Mode Terminals*
- EPI (*external presentation interface*) für AIX- und OS/2-Clients
  - Ver-/Entpacken der 3270-Datenströme
  - ggf. Konvertierung zwischen ASCII und EBCDIC
- analog FEPI (*front end programming interface*) für Mainframe CICS
- ECI (*external call interface*, s.u.) zum Aufruf von CICS-Programmen von außerhalb

➪ BMS (*basic mapping support*): *Forms Manager*



# CICS (4)

❑ Kommunikation




➪ Transaction Routing (s.o.)

➪ EPI (s.o.)

➪ MRO (*multiregion operation*) und ISC (*intersystem communication*) zur Kommunikation zwischen *CICS Regions*, die auf demselben Mainframe laufen

➪ DTP

- *distributed transaction processing*
- eigentlich APPC-Interface von LU6.2 für Peer-to-Peer
- unterstützt Datenkonvertierung
- schwierig (Sessions, Programzustände, ...)
- ermöglicht Zusammenarbeit mit transaktionalen, LU6.2-fähigen Applikationen, die außerhalb der *CICS Region* laufen (erklärt Popularität des LU6.2 Gateways in anderen TP-Monitoren)

# CICS (5)

❑ Kommunikation (Forts.)

➩ DPL (*distributed program link*, synchroner RPC)

- in der Nutzung einfacher als DTP
- weniger flexibel
- unterstützt Datenkonvertierung

➩ ECI (s.o.)

```
An ECI call expressed in COBOL:
Call '_CICS_ExternalCall'
USING BY REFERENCE ECI-PARMS
RETURNING ECI-RETURN-CODE.
```

```
An ECI call expressed in C:
ECI_PARMS          EciBlock;
cics_ushort_t      Response;
:
:
Response = CICS_ExternalCall (&EciBlock);
```

- ähnlich DPL, nur für außerhalb einer *CICS Region*
- wird in IBM's CORBA-OTS-Implementierung zur Kommunikation von CORBA-Objekten mit CICS-Applikationen genutzt

➩ verschiedene Möglichkeiten des *Queued TP*

- *asynchronous processing*: nicht-persistent; z. B., um Request zu bestimmter Zeit zu bearbeiten
- *transient data*: nicht-persistent; als Eingabe für Terminals, Drucker, etc.
- *temporary storage*: persistent



# CICS (6)

❑ DB-Zugriff

➩ frühere Mainframe-Versionen

- Applikationen mussten DB-Operation an CICS richten wegen dem Blocking-Problem (beachte: TP-Monitor-Threads)
- klassisch: Zugriff auf VSAM-Dateien (*Virtual Sequential Access Method*)
- unter den ersten TP-Systemen, die *remote data access* unterstützten (*function shipping*: Zugriff auf entfernte VSAM-Datei)

➩ heute: DTP X/OPEN
(zumindest AIX- und UNIX-Versionen von CICS)

# CICS (7)

❑ System Management

➪ Funktionen

- Starten/Stoppen von Regions
- Steuerung der *Transaction Policy* (z. B. Start der Transaktion bei Eingabe am Terminal oder bei Erhalt der Nachricht, etc.)
- *Region Recovery*
- *Auditing*
- *Batch job scheduling*
- *Accounting*
- *(Performance) monitoring*

➪ Funktionen werden als *system transactions* abgewickelt zur Isolation von Transaktionsprogrammen, die auf denselben Ressourcen arbeiten

➪ *Master terminal*

- Kontrolle mehrerer Regions im CICS System
- Entgegennehmen/Behandeln von Nachrichten von Programmen
- *Enable/Disable*
  - CICS-Dateien
  - Transaktionen
  - Kommunikation zwischen Regions
  - Einschalten des Debug-Trace
  - Kopien von Transaktionsprogrammen installieren



# CICS (7)

❑ Transaktionsprogramme

➪ Beispiel:

```
PROCEDURE DIVISION.

001-MAIN SECTION.
        EXEC CICS HANDLE CONDITION
                LENGERR    (001-LENGTH-ERROR)
                ERROR      (001-GENERAL-ERROR)
        END EXEC.
        PERFORM    001-SEND.
        PERFORM    001-RECEIVE.
        PERFORM    001-DEBIT.
        PERFORM    001-CREDIT.
        PERFORM    001-SUCCESS.
        EXEC CICS RETURN.
        END EXEC.

001-SEND SECTION.
        EXEC CICS SEND
                MAP ('MAPDBCR')
                MAPSET ('DBCRSET')
                MAPONLY
                ERASE
        END EXEC.

001-RECEIVE SECTION.
        EXEC CICS RECEIVE
                MAP ('MAPDBCR')
                MAPSET ('DBCRSET')
                INTO (INPUT-DBCR-AREA)
        END EXEC.
```

# CICS (8)

- Transaktionsprogramme (Forts.)
  - Beispiel (Forts.):

```
001-DEBIT SECTION.
          MOVE INPUT-CUST-NO TO CUST-NO.
          MOVE DB-ACCT-NO TO ACCT-NO.
          MOVE INPUT-AMT TO AMOUNT.
          MOVE ‘DEBITPGM’ TO PGM-NAME.
          EXEC CICS LINK
                    PROGRAM (PGM-NAME)
                    COMMAREA (PGM-COMAREA)
                    LENGTH (24)
          END EXEC.
          MOVE BALANCE TO DB-ACCT-BAL.
001-CREDIT SECTION.
          MOVE CR-ACCT-NO   TO ACCT-NO.
          MOVE ‘CREDITPGM’  TO PGM-NAME.
          EXEC CICS LINK
                    PROGRAM (PGM-NAME)
                    COMMAREA (PGM-COMAREA)
                    LENGTH (24)
          END EXEC.
          MOVE BALANCE TO CR-ACCT-BALL.
001-SUCCESS SECTION.
          MOVE ‘ACCOUNT BALANCES AFTER THE
                    WITHDRAWAL ARE: ‘
                    TO TXFR-MESSAGE.
          EXEC CICS SEND
                    MAP (‘MAPDBCR’)
                    MAPSET (‘DBCRSET’)
                    FROM (OUTPUT-DBCR-AREA)
                    DATAONLY
          END EXEC.
```



# CICS (9)

- Transaktionsprogramme (Forts.)
  - Beispiel (Forts.):

```
001-LENGTH-ERROR SECTION.
          MOVE INVALID ACCOUNT NUMBER LENGTH,
                    PLEASE TRY AGAIN ‘
                    TO TXFR-MESSAGE.
          EXEC CICS SEND
                    MAP (‘MAPDBCR’)
                    MAPSET (‘DBCRSET’)
                    FROM (OUTPUT-DBCR-AREA)
                    DATAONLY
          END EXEC.
          TOTO 001-MAIN.
001-GENERAL-ERROR SECTION.
          MOVE ‘TRANSFER OPERATION NOT AVAILABLE
          AT THE TIME, PLEASE
          TRY AGAIN LATER’
          TO TXFR-MESSAGE.
          EXEC CICS SEND
                    MAP (‘MAPDBCR’)
                    MAPSET (‘DBCRSET’)
                    FROM (OUTPUT-DBCR-AREA)
                    DATAONLY
          EXEC EXEC.
          EXEC CICS RETURN.
          END EXEC.
```

# CICS (10)

❑ Transaktionsprogramme (Forts.)

➪ Beispiel (Forts.):

- Pre-Compiler zur Übersetzung der CICS-Anweisungen (im Beispiel fett gedruckt) im Programm
- das Beispiel verdeutlich die wichtigsten CICS-API-Elemente bei Bearbeitung eines Requests innerhalb eines Transaktionsprogramms
- CICS unterstützt jedoch auch sehr viel reichhaltigere Applikationsstrukturen (s.o.)

➪ *Transaction Definition*

- beinhaltet
  - Sicherheitsinformation
  - Priorität
  - Ortsinformation
- sowohl für First-Level-Programme als auch für Second-Level- (d. h., über LINK, ECI oder DPL aufgerufene) Programme



# TUXEDO (1)

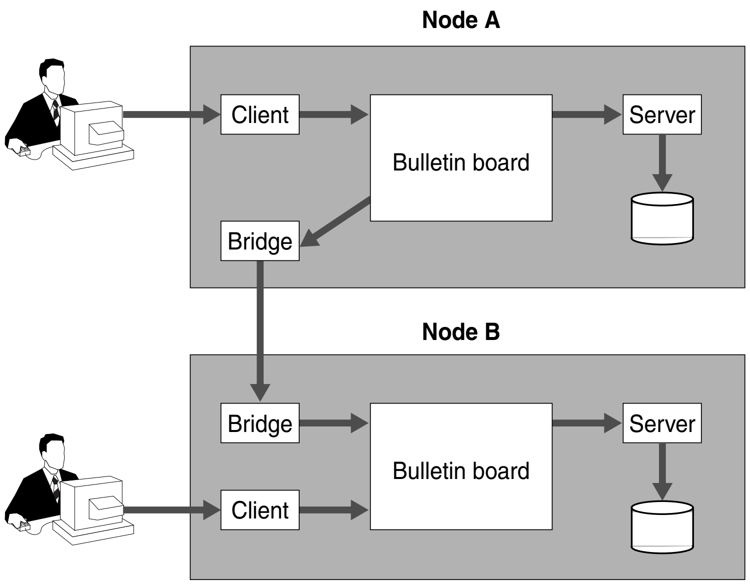
❑ Allgemeines

➪ BEA Systems

➪ für mehr als 15 Betriebssysteme, insbesondere UNIX Plattformen und Windows NT

➪ bemerkenswert: beeinflusste X/OPEN-DTP-Standards maßgeblich

❑ Systemarchitektur



# TUXEDO (2)

❑ Systemarchitektur (Forts.)

➪ API: ATMI

- *Application to Transaction Manager Interface*
- Kollektion von Laufzeitdiensten mit Unterstützung von
  - Kommunikation
  - verteilte Transaktionen
  - System-Management
- nutzt zugrundeliegende Betriebssystem- und DBMS-Dienste viel stärker als die 15 Jahre früher entstandene CICS API

➪ *Bulletin Board*

- enthält, ähnlich wie *CICS Tables*, Konfigurationsinformation für viele TP-Monitor-Funktionen, z. B.:
  - Namen von Transaktionsdiensten
  - Abbildung dieser Namen auf *Transaction-Server*-Adressen
  - Information für *parameter-based routing*
  - Konfigurationsoptionen für *Transaction Server*
- ein *Master Bulletin Board*
- Bulletin Board eines Knotens wird in *Shared Memory* geladen
- dynamisches Hinzufügen, Löschen und Ändern von *Services* und *Servern*



# TUXEDO (3)

❑ Systemarchitektur (Forts.)

➪ TUXEDO-System besteht aus Client- und Server-Prozessen

- Clients
  - Presentation Servers
  - dürfen Transaktionen starten
- Server (*System/T Package*)
  - Zugriff auf transaktionale RM
  - kann *Transaction Server* oder *Workflow-Controller* oder beides sein!

➪ *Domain*

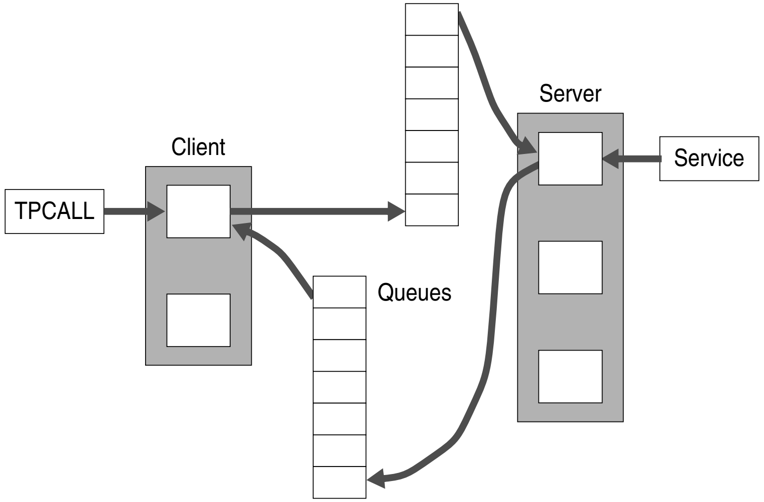
- beschreibt *Scope* der Computer, die an der Verarbeitung teilnehmen können
- vgl. *Cell*-Konzept von Encina (s.u.)

➪ *Service*

- Name eines *Service-Interface*
- Service kann von mehreren *Transaction Servers* unterstützt werden und umgekehrt
- Abbildung von Services auf *Transaction Server* ist im *Bulletin Board* vermerkt
- bei Aufruf eines Services
  - ‘forwarded’ *Bulletin Board* diesen Aufruf zu einem *Transaction Server* weiter;
  - Nutzung einer *Bridge* bei *Remote Servers*

# TUXEDO (4)

- Systemarchitektur (Forts.)
  - Request Message Flow




  - Transaktionskontrolle
    - explizit über API-Primitive (*tpbegin, tpcommit, tpabort*)
    - automatisch über Flags in Programm bzw. Konfigurationsdatei
      - Transaktion wird automatisch bei Programmaufruf gestartet und bei Rückgabe der Kontrolle wieder geschlossen
      - falls Transaktion bereits im Client gestartet wurde, nimmt Programm an dieser Transaktion teil
    - asynchron



# TUXEDO (5)

- Presentation Server
  - *Sytem/WS Package* für *Presentation Server*, die TUXEDO-Server von PC-Clients aus zugreifen (Client/Server 4GLs)
  - Alternative: DES (*Data Entry System*) für Masken und Menus
    - neben Service-Name und Eingabedaten umfasst Maskeneingabe Flags, die bspw. zur Transaktionskontrolle oder automatischem Retry nach BS-Interrupt dienen
  - Fehlerbehandlung auf der Applikationsebene
  - System-Server
    - BQ (*background jobs; batch emulation*)
    - FRMRPT (druckt Kopie einer Dateneingabemaske)
    - AUTHSVR (Client-Authentifizierung)
    - ...
- Kommunikation
  - TUXEDO unterstützt auf der Basis von Peer-to-Peer-Protokollen
    - Peer-to-Peer (*conversational*)
    - RPC (synchron, asynchron)
    - *Event posting* über *bulletin board*
    - *persistent message queuing (System/Q)*

# TUXEDO (6)

- Kommunikation (Forts.)
  - Event System
    - mögliche Reaktionen (optional transaktional)
      - Aufruf eines Services
      - Nachricht in Warteschlange schreiben
      - Nachricht in Datei schreiben
  - keine Kontexte über mehrere RPCs hinweg, nur durch Message-Passing
  - Server kann beim Aufruf eines anderen Servers festlegen, ob der Aufgerufene innerhalb oder außerhalb der Transaktion des Aufrufers laufen soll
  - Flexible Möglichkeiten der Weitergabe der Kontrolle (z. B. A ruft B auf, B ruft C auf, C gibt Kontrolle an A zurück)
  - Gateway nach CICS/MVS via APPC/LU6.2; jedoch kein Syncpoint Level 2
  - System/Q
    - statt Nutzung der Eingabe-Queue eines Servers kann Nachricht in eine persistente Queue gelegt werden
- DB-Zugriff
  - 2PC durch Built-In-TA-Mgr
  - Unterstützung aller XA-fähigen RM (z. B. Oracle, Sybase, Ingres, Informix, etc.)



# TUXEDO (7)

- System Management
  - TMIB (TUXEDO Management Information Base)
    - Get/Set-Operationen auf Konfigurationsinformation
    - grafisches Interface
    - Kontrolle der Systemkonfiguration über den *Master Node* des *Bulletin Boards*
      - Start-up/Shut-down
      - Kompilation der Konfigurationsdatei
      - Laden des *Bulletin Boards* von der Konfigurationsdatei
      - Hinzufügen/Löschen von Services
      - Verschieben von Servern zwischen *Nodes*
      - Änderung von Server-Prioritäten
      - Unterstützung eines *backup master nodes*
      - ...
  - Sicherheit
    - extern: z. B. durch BS
    - optional: applikationsspezifische Authentifizierungs- und Autorisierungsmechanismen
  - Parameter-based Routing
  - Lastbalancierung
    - Nutzung von Statistiken

# TUXEDO (8)

- Programmierung
  - Beispiel
    - Client-Programm, das ATMI nutzt:

```
main ()
{
char    *rbuf;
char    *reqfb;
long    reqlen, r1len, r2len;
. . . / / construct reqfb, which contains parameters to the call to DBCR
reqfb=tpcalloc(„FML“, „“, 200); /* allocate typed buffer for request */
tpcall („DBCR“, (char *)reqfb, 0, (char **)&reqfb, (long *)&reqlen,
        TPSIGRSTRT);
. . . / / construct the input buffer, rbuf, for the next call
tpcall („PRINTER“, rbuf, r1len, &rbuf, &r2len, TPNOTRAN)
}
```



# TUXEDO (9)

- Programmierung
  - Beispiel
    - Server-Programm, das ATMI nutzt:

Request-Information:
Service-Name
Flags (z. B. Transaktionsmodus ja/nein)
Parameter/Längenangaben
Connection-Deskriptor (falls conv.)
Application Authentication Key
Client-ID (ermöglicht *tpnotify*)

```
void DBCR (TPSVCINFO *buf);
{
. . .
tpbegin ()
ret1=tpcall        („DEBIT“, (char *)reqfb, 0, (char **)&reqfb, (long *)&reqlen,
                   TPSIGRSTRT);
. . . / / check for errors and construct next request
ret2=tpcall        („CREDIT“, (char*)reqfb,0, (char **)&reqfb, (long *)&reqlen,
                   TPSIGRSTRT);
. . . / / check for errors
if (ret1 == -1 | | ret2 == -1) tpabort (); else tpcommit ();
. . . / / construct the reply buffer
tpreturn (TPSUCCESS, 0, buf ->data, sizeof (struct aud), 0);
}
```

# TUXEDO (10)

- ❑ Programmierung
  - ➪ Eintragung/Registrierung der Services über Service-Template
  - ➪ Main (von System/T realisierte Prozedur)
    - Schleife, die jeweils die Eingabeschlange für Requests prüft
    - ggf. Aufruf der Sub-Routine, die den aufgerufenen Service implementiert
    - sendet Reply, nachdem sie Kontrolle von der Sub-Routine zurückbekommen hat
    - ruft bei Shut-Down Cleanup-Routine
  - ➪ Weitere ATMI-Funktionen
    - asynchroner Aufruf: (statt *tpcall*) *tpacall* und *tpgetreply*
    - *tpgetlev*: zeigt an, ob Sub-Routine im Transaktionsmodus läuft
    - *tpcancel*
    - *tpgprio* (*get priority*)
    - *tpsprio* (*set priority*)
  - ➪ optional: Nutzung von X/OPEN TX
    - Unterschied zu ATMI:
      TX unterstützt *chained,* ATMI *unchained*

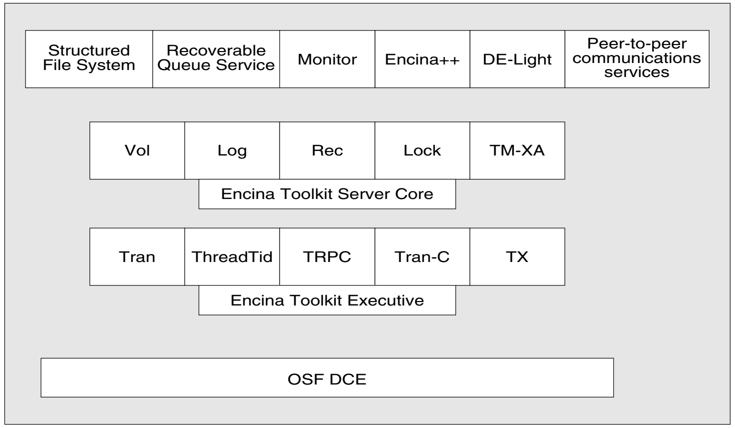


# Encina (1)

- ❑ Allgemeines
  - ➪ seit 1989 von Transarc, die 1994 von IBM übernommen wurden
  - ➪ für verschiedene UNIX-Plattformen und Windows NT (Clients auch auf Windows und OS/2)
  - ➪ Implementierung der folgenden Standards: X/OPEN XA, TX, TxRPC und OMG OTS
- ❑ Systemarchitektur (Übersicht: siehe nächste Folie)
  - ➪ Toolkit
    - *enabling technology*, die auch in CICS für UNIX und Digital's ACMxp genutzt wird
    - bestehend aus verschiedenen Komponenten mit OSF DCE als Basis für Betriebssystem- und Maschinen-unabhängige Kommunikation
    - *Encina Toolkit Executive*:
      Basisdienste für alle Prozesse in der Encina-Umgebung
      - *Tran*: erlaubt es Prozessen, verteilte Transaktionen zu erzeugen, daran teilzunehmen und sie zu beenden (Bibl.)
      - *ThreadTid*: Zuordnung von Transaktionskontexten und Threads (Bibl.)
      - *TRPC*: unterstützt TIDL, das wiederum eine Erweiterung der DCE-RPC-IDL hinsichtlich von TA-Kontexten ist (Bibl. *und Compiler)*
      - *Tran*-C: Makros/Laufzeitsystem für Transaktionskontrolle und Exception Handling in C
      - *TX*: nach X/Open

## Encina (2)

❑ Systemarchitektur (Forts.): Übersicht






## Encina (3)

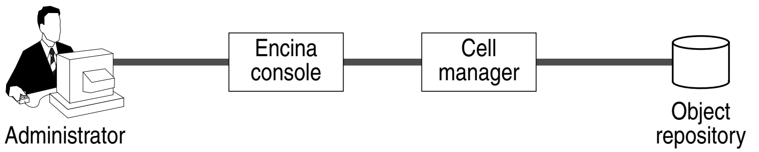
❑ Systemarchitektur (Forts.)

➪ Toolkit (Forts.)

- *Encina Toolkit Server Core*
  Module zur Erzeugung von (ACID-)Servern
  - *Vol*: Verwaltung von Spiegelplatten
  - *Log*: *append-only* Speicherungsmodul zur Protokollierung von transaktionalen Modifikationen und Zuständen
  - *Rec*: *recoverable* Speicherdienst mit WAL, der Zustand nach einem Fehler wiederherstellen kann
  - *Lock*: *(2P-) Locking Manager* mit konventionellen *Read/Write/Intension-Locks* für geschachtelte Transaktionen
  - *TM-XA*: Kommunikation nach X/Open DTP zwischen RMs und dem *Encina Transaction Service*
- *Structured File System* (SFS)
  - unterstützt TRPC und geschachtelte Transaktionen
  - verschiedene Organisationsformen, z. B. B*-Baum
  - verschiedene Möglichkeiten für Sekundärindizes
  - Isolationsgrade: *browse, cursor stability, repeatable read*
- *Recoverable Queue Service* (RQS)
  - unterstützt TRPC und geschachtelte Transaktionen
  - FIFO, Prioritäten, Zugriff über Schlüssel

# Encina (4)

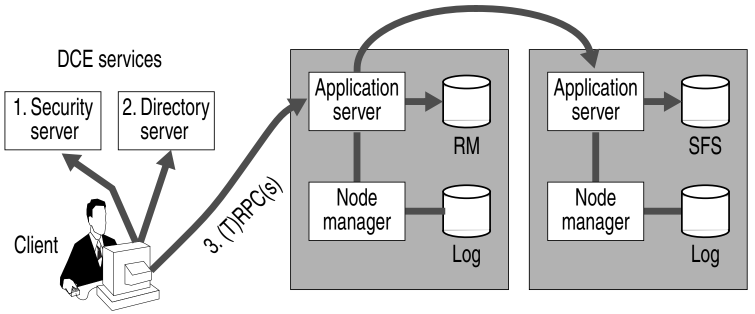
- Systemarchitektur (Forts.)
  - Toolkit (Forts.)
    - *Peer-to-Peer Communications* (PPC)
      - SNA LU6.2, Syncpoint Level 2
      - wird genutzt, um Mainframe-Transaktion mit Encina-Transaktion in einer Transaktion zusammenzufassen
    - *DE-Light*
      - zum Aufruf von Encina-Services von (kleineren) Clients (Bibl.), die unter Windows oder in Java-Interpretern laufen
      - dynamischer Aufruf von Encina-Applikations-Servern mittels TRPC und Nutzung von TX
  - Monitor
    - *Cell*
      - administrative Einheit zur Verwaltung von Ressourcen
      - *Cell Manager*: Verwaltung der Konfiguration
      - *Node Managers*: Monitoring und Verwaltung von Ressourcen auf einer Maschine; Start/Restart/Stop von Prozessen; führt Log für *recoverable* Applications-Server
      - Konfigurationsdaten in *Object Repository*, das Klassenhierarchien/Vererbung zur Gruppierung der Eigenschaften verfügbarer Server nutzt






# Encina (5)

- Systemarchitektur (Forts.)
  - Encina Clients und Client/Server-Kopplung
    - *Encina Client Library*
      - integriert Präsentationsdienste wie *Visual Basic, Visual C++, PowerBuilder*
      - direkte (RPC-)Verbindung mit Applikations-Server
      - Sicherheit
      - Caching
    - Encina++
      - DCE IDL
      - Transarc's Transactional IDL
      - OMG OTS
    - Verbindungsaufnahme und synchrone Kommunikation oder asynchron über RQS
    - ggf. mehrere TRPCs auf verschiedene AS in derselben TA
    - einfache Lastbalancierung durch gewichtete, zufällige Auswahl der Server



# Encina (6)

❑ Transaction Server

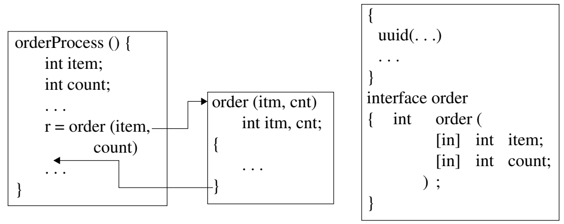
➩ verschiedene APIs nutzbar, die jeweils auf TRAN abgebildet werden

| | |
|---|---|
| . . .<br>tx_begin ();<br>. . .<br>tx_commit (); | X/Open TX API supports chained and unchained transactions. Transarc extensions support nested transactions, too. |
| Transaction{<br>. . . */ }<br>transaction { /* sub transaction<br>onAbort { /* compensate without<br>aborting top-level transaction. */ }<br>. . .<br><br>. . . | Tran-C and Tran-C++ extend the C and C++ programming languages through C macros and C++ class libraries. They support nested transactions. |
| Commit ();<br>. . . . .<br>Commit (); | CPI-RR is part of IBM's SAA, providing interoperability with mainframe systems. It is a nonnested chained transaction API: |



# Encina (7)

❑ Programmiermodelle

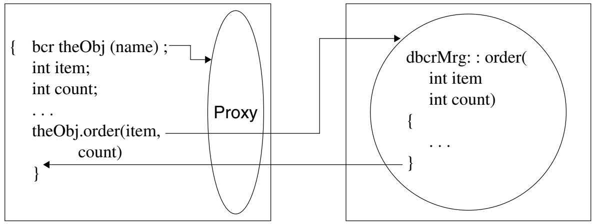



Order.tidl

***Basic RPC***. The *TIDL compiler* uses hidden arguments to send the transaction context with the marshaled data. Using explicit binding, the client selects the server instance directly. Using transparent binding, the monitor selects a server instance based on administrator-set criteria.

# Encina (8)

❑ Programmiermodelle (Forts.)

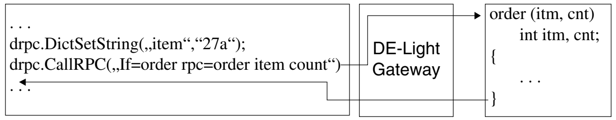



*Encina++* supports invocation of object interfaces with four ***binding options***: simple, where binding is to any object supporting the called interface; reference, where binding is based on a previously obtained reference, frequently from factory objects (which create instances); named object, where binding is to an object registered with the monitor; and named server, where binding is to a conventional server registered with the monitor.

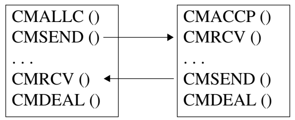


# Encina (9)

❑ Programmiermodelle (Forts.)




***Dynamic RPC*** invocation using *DE-Light* RPC invocation. In this model, no client stubs are used; instead, the DE-Light gateway interprets IDL/TIDL files.




*CPI-C* is IBM's ***peer-to-peer*** calling convention for transactional interoperability with mainframe systems.

# Encina (10)

- DB-Zugriff
  - ➩ neben SFS und RQS können Applikationen XA-fähige RMs aufrufen
  - ➩ Encina TM-XA bietet zusätzlich geschachtelte Transaktionen
- System Management
  - ➩ *Object Repository*
    - Konfigurationsdaten und Laufzeit/Zustands-Daten
      - Anzahl der Threads pro Prozess
      - Anzahl der Prozesse pro Machine
      - Start-/Stop-Zeiten von Prozessen
      - Audit- und Trace-Daten
    - Sicherheit
      - Client-Authentifikation
      - Verschlüsselung
      - Data Privacy
      - Beschränkung des Zugriffs auf Server
      - *Access Control Lists* für Applikations-Server, Interfaces und einzelne RPCs



# Encina (11)

- Programmierbeispiel
  - ➩ dbcr.tidl in Encina++:

```
[     uuid (8c2a59a4-c5d3-11ce-828a-9e621218aa77),
      version (1.0),
      exceptions (invalid_account, illegal_amount)
]
interface dbcr
{     void withdraw (  [in]        long account,
                       [in]        long amount);

      void deposit (   [in]        long account,
                       [in]        long amount);

      [transaction_optional] long query [in] long account);
}
```

# Encina (12)

- Programmierbeispiel (Forts.)
  - Client.C in Encina++:

```
#include <ots/otsClient.H>
#include „dbcrTC.H“

void DBCR
(char *nameOfObjToUse, long from, long to, long amount)
{     dbcr theObject (nameOfObjToUse) ;

      transaction    {    theObject.withdraw(from, amount);
                          theObject.deposit (to, amount) ; }
      onCommit       {    cout << „Transfer succeeded“
                          << end1; }
                     {    cout << „Transfer failed because “
                          << abortReason ()
                          << end1; }
}

main (int argc, char **argv)
{     OtsClient      theClient;
      long           from, to, amount;

      cout << „Withdraw from account? “;
      cin >> from;
      cout << „Deposit to account? “;
      cin >> to;
      cout << „Amount? “;
      cin >> amount;
      DBCR (argv[1], from, to, amount);

}
```



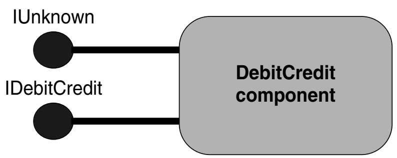
# Encina (13)

- Programmierbeispiel (Forts.)
  - Server.C in Encina++:

```
#include <ots/otsServert.H>
#include „dbcrTS.H“

main (int argc, char **argv)
{     OtsServer the Server;
      dbcrMgr theObject (argv[1]);
      theServer.RegisterResource (. . .);
      theServer.Listen ();
      return (0);
}

void dbcrMgr : :withdraw (idl_long_int account, idl_long_int amount )
{     if (amount <= 0) throw illegal_amount ();
      EXEC SQL
      . . .
      END EXEC
}

void dbcrMgr : :deposit (idl_long_int account, idl_long_int amount)
{     if (amount <= 0) throw illegal_amount ();
      EXEC SQL
      . . .
      END EXEC
}
```

# MTS (1)

- ❏ Allgemeines
  - ➪ Microsoft Transaction Server, 1996/97
  - ➪ Komponenten-basiert, objektorientiert
  - ➪ Windows
  - ➪ XA-fähige TM und RM
- ❏ (D)COM
  - ➪ sprachunabhängiger Binärstandard für die Kommunikation zwischen Komponenten
  - ➪ Komponente
    - ausführbarer Objekt-Code
    - kann mehrere Interfaces haben




    - jedes Interface ist eine Kollektion von Methoden
  - ➪ Objekt
    - Instanz einer Komponente
    - *Class Factory*
  - ➪ Interface-Spezifikation
    - Methoden-Signaturen
    - IID: global eindeutiger Identifikator (128 Bit)
    - unveränderbar
    - jede Komponente bietet Interface *IUnknown*
      - Methode *QueryInterface* erlaubt Abfrage, ob ein Objekt ein bestimmtes Interface (IID) unterstützt



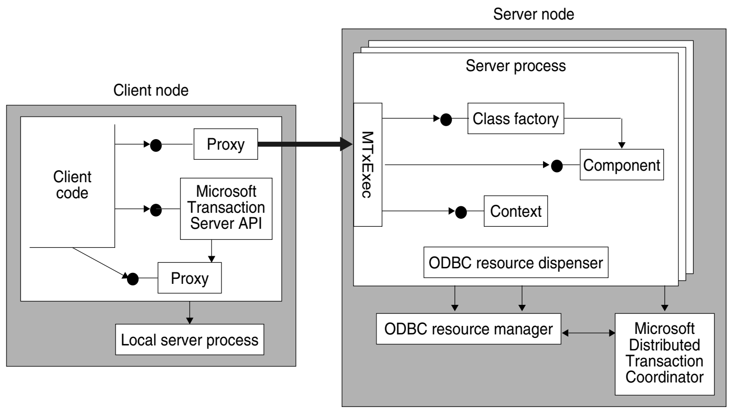
# MTS (2)

- ❏ Systemarchitektur
  - ➪ eine *Microsoft Transaction Server Application* besteht aus:

| Type of Object | Shares Data? | Durable Data? | Description |
|---|---|---|---|
| Client | Optional | Optional | Application code in the presentation server to interact with end users via a 4GL, terminal interface, Internet browser, and so on. |
| MTx object | No | No | Application business logic for workflow control and/or a transaction server. It consists of methods in a single-user nonpreemptive component with private member data (not shared b other objects) and is therefore easy to program. |
| Resource dispenser | Yes | No | Providers of shared nondurable resources, such as ODBC connections or memory heaps, which are shared by objects in a single process. Resource dispensers require more skill to implement than MTx objects as they involve sharing and, hence, require synchronization to provide correct behavior. |
| Resource manager | Yes | Yes | Resource managers, such as Microsoft SQL Server, provide shared durable state, participate in two-phase commit, and provide isolation. |

## MTS (3)

❏ Systemarchitektur (Forts.)






## MTS (4)

❏ Systemarchitektur (Forts.)

➪ Deployment/Verwaltung von Komponenten erfordert Komponentenspezifikation

- Dynamic Link Library (*.dll*)
  - dieselbe Komponente kann sowohl im Prozess des Erzeugers (Objekt oder Client) laufen als auch in separatem Prozess, der wiederum auf demselben *node* oder *remote* sein kann
- Definitionen der *Interfaces*
- Konfigurationsinformation, z. B.transaktionales Verhalten
  - *not supported*
  - *required*
  - *requires new*

➪ *MTx objects*

- laufen in von MTS verwalteten Server-Prozessen
- DCOM ist verantwortlich für die Ortstransparenz (vgl. CORBA)
- *MTxExec*
  - propagiert Transaktionskontexte
  - verwaltet Threads und deren Zuordnung innerhalb eines Server-Prozesses

# MTS (5)

- Presentation Server
  - Applikationen
    - Client-Applikation auf einem PC
    - HTML-basierte Applikation in einem Browser
    - in HTML-Seiten eingebettete Active-X-Komponente
  - Desktop-Komponenten können Applikationen über http ansprechen
  - Nutzung von COM-APIs zum Erzeugen von und Zugriff auf Server-Objekte
- Kommunikation
  - Methodenaufrufe als objektorientierte Form des RPC
  - Ortstransparenz
  - synchroner Methodenaufruf und mittlerweile auch *Queued*
  - *MS DTC: Built-in-Transaction-Manager*
- System Management
  - *Server Group*
    - Menge von *Nodes* als Einheit der Verwaltung
    - gemeinsamer Katalog
    - Server-Klassen zur Lastbalancierung (über mehrere Knoten)
  - *Package*
    - zur Gruppierung von Komponenten (*unit of trust*)



# MTS (6)

- DB-Zugriff
  - RMs, die mit MS-DTC über OLE-Transaktionen oder XA interoperieren können
- Programmierbeispiel
  - Debit-Credit-Client

```
// All the #include statements are omitted for readability

HRESULT DBCRclient (long from, long to, long amount)
{    HRESULT hr = S_OK;
     CLSID TransferMoneyClsid;
     ITransferMoney* pTransferMoney = NULL;


     // Create an object of the Transfer server component
     CLSIDFromProgID ( L“TransferMoney.Server.1“,
                       &TranferMoneyClsid);
     CoCreateInstance (  TransferMoneyClsid, IID_IDBCR,
                       (void**) &pTransferMoney);

     // Transfer calls debit and credit within the same transaction
     hr = pTransferMoney->Transfer (from, to, amount);

     // Release the TransferMoney object and return
     pTransferMoney->Release ();
     return hr;
}
```

# MTS (7)

- Programmiereispiel (Forts.)
  - TransferMoney-Server

```
STDMETHODIMP
TransferMoney : :Transfer (IN long from, IN long to, In long amount)
{    HRESULT hr;
     IObjectContext* pContext = NULL;
     IDBCR* pAccount = NULL;

     / / Get reference to my context
     GetObjectContext (&pContext) ;

     / / Create the DBCR object via my context and transaction
     CLSIDFromProgID (L"DBCR.Server.1", &DBCRClsid) ;
     pContext->CreateInstance(DBCRC1sid, IID_IDBCR,
                              (void**) &pAccount) ;

     / / Use the same reference to call 2 different objects via
     / / „just-in-time activations"
     pAccount->initNew (from) ;
     hr = pAccount->debit (amount) ;
     if   (SUCCEEDED (hr) )
     {    / / reuse the same object for a different account
          pAccount->initNew (to) ;
          hr = pAccount->credit (amount) ;
     }

     / / The transaction automatically commits or aborts upon return
     if (SUCCEEDED (hr) ) pContext->SetComplete () ;
     else pContext->SetAbort () ;
     return hr ;
}
```



# MTS (8)

- Programmiereispiel (Forts.)
  - Debit-Server

```
HRESULT DBCR::debit(IN long amount)
{    HRESULT hr;
     IObjectContext* pContext = NULL

// get reference to my context
GetObjectContext(&pContext);

if (amount <= 0)
     hr = E_INVALIDARG;
else {
     // do the requested work here
     hr = S_OK;
}

if (SUCCEEDED(hr))
     // This object is done for now and agrees to commit
     pContext->SetComplete();
else
     // This object is done for now and will not agree to commit
     pContext->SetAbort();
return hr;
}
```

# MTS (9)

❑ Programmiereispiel

➪ Interface-Definition für Debit-Credit-Server

```
[   object
    uuid(27870601-E1A7-11CF-B899-0080C7394688),
    dual,
    helpstring(„IDBCR Interface“),
    pointer_default(unique)
]

interface IDBCR : IDispatch
{   import      „oaidl.idl“;
    HRESULT  initNew(     [in] long Account);
    HRESULT  debit(       [in] long Amount);
    HRESULT  credit(      [in] long Amount);
};

[   uuid(27870600-E1A7-11CF-B899-0080C7394688),
    version(1.0),
    helpstring(„DRCR 1.0 Type Library“)
]
library DBCRLib
{   importlib(„stdole32.tlb“);

    [           uuid(27870605-E1A7-11CF-B899-0080C7394688),
                helpstring(„DBCR Class“)
    ]
    coclass DBCR
    {
                [default] interface IDBCR;
    };
};
```



# Zusammenfassung (1)

❑ Die hier vorgestellten TP-Monitore

➪ folgen dem vorgestellten 3-Tier-Modell

- *Presentation Server*
- *Workflow Controller*
- *Transaction Server*

➪ unterstützen Zugriff von modernen Desktop-Geräten

➪ bieten unterschiedliche APIs

➪ alle, außer MTS, für UNIX und Windows NT

➪ unterstützen alle X/Open XA

➪ können alle, außer MTS, miteinander über LU6.2 interoperieren

❑ Zusatzinfo (vgl. Kapitel 5)

➪ die in diesem Kapitel vorgestellten TP-Monitore weisen alle einen ‘schwachen’ *Workflow Controller* auf,

➪ d. h., es wird keine Applikations-Struktur außerhalb der *Transaction Server* definiert und kontrolliert

➪ nur wenige TP-Monitore kontrollieren Applikationsstrukturen (im Sinne eines Workflow-Management)

➪ Beispiele

- ACMxp (Digital) nutzt
  STDL (*structured transaction definition language*)
- auch Pathwas/TS (Tandem) unterstützt Applikationsstrukturen

## Zusammenfassung (2)

- ❑ CICS
  - bekanntester, erster TP-Monitor
  - Kommando-basiertes API
  - hauptsächlich Peer-to-Peer, aber auch RPC und Queued
- ❑ TUXEDO
  - führender TP-Monitor für UNIX, auch Windows NT
  - einfaches Service-basiertes API
  - flexible Kommunikationsmöglichkeiten
- ❑ Encina
  - basiert auf dem modularen Encina Toolkit
  - TRPC basierend auf OSF DCE, aber auch Peer-to-Peer und RQS
  - geschachtelte Transaktionen
  - unterstützt Implementierung des OMG OTS
  - C++-Klassenbibliothek
- ❑ MTS
  - basierend auf (D)COM und OLE, d. h., Integration von Komponenteninfrastruktur mit TP-Monitor
  - beinhaltet separat zugreifbaren Transaktions-Manager MS-DTC, der neben nativem MS-API auch XA unterstützt



## Zusammenfassung (3)

- ❑ allgemeine Entwicklung
  - Mainframe
  - Verteilung (Unix, Windows NT)
  - Komponenten(infrastrukturen)
  - WWW (Integration in E-Business-Plattformen)