Transaktionale Informationssysteme

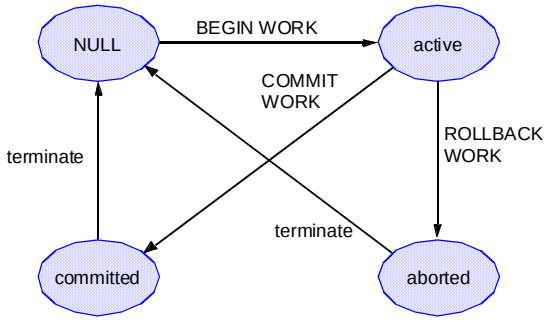
-

6. (Erweiterte) Transaktionsmodelle

Norbert Ritter
Datenbanken und Informationssysteme
vsis-www.informatik.uni-hamburg.de

Beschreibung von TA-Modellen (1)

- Zustands-/Übergangsdiagramm
  - für flache Transaktionen aus der Sicht der Anwendung
  - Erweiterbar?



Beschreibung von TA-Modellen (2)

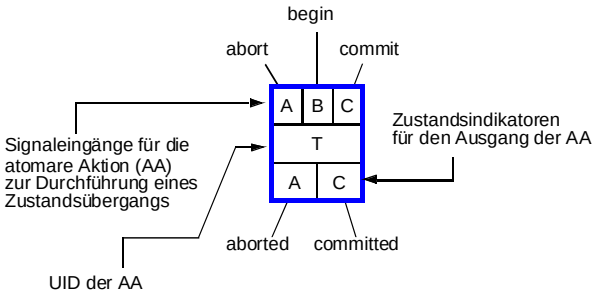
- Abhängigkeiten zwischen Transaktionen
  - Strukturelle Abhängigkeiten
    - beschreiben hierarchische Organisation des Systems als abstrakte Datentypen zunehmender Komplexität
    - festgelegt, da 'Aufrufhierarchie im Code vorgegeben'
  - Dynamische Abhängigkeiten
    - durch Nutzung von gemeinsamen Daten
    - können eine beliebige Anzahl sonst unabhängiger atomarer Aktionen einhüllen
- Strukturelle und dynamische Abhängigkeiten als Regeln beschreibbar!

Beschreibung von TA-Modellen (3)

- Regelaufbau
  - Aktiver Teil
    - BEGIN WORK (B), ROLLBACK WORK (A, abort) und COMMIT WORK (C) verursachen Zustandsänderungen einer atomaren Aktion
    - Transaktionsmodelle können Bedingungen definieren, die diese Events auslösen
  - Passiver Teil
    - Abhängige atomare Aktionen dürfen bestimmte Zustandsübergänge nicht von selbst durchführen
    - Es sind Regeln erforderlich, die die Bedingungen für diese Zustandsübergänge spezifizieren

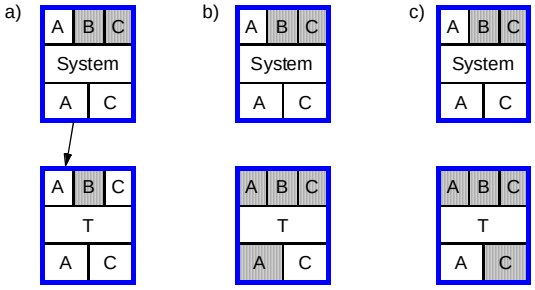
Beschreibung von TA-Modellen (4)

- Grafische Notation
  - Zur graphischen Darstellung eines Transaktionsmodells wird jede Instanz einer atomaren Aktion (AA) als eine ‚Box' veranschaulicht
  - Schattierte Bereiche bezeichnen verbotene Ereignisse oder Ausgänge!



Beschreibung von TA-Modellen (5)

- System-TA beschreibt Ablaufumgebung
  - immer aktiv, solange System "läuft"
  - führt kein Commit durch
  - führt Abort nur als Ergebnis eines System-Crashs durch
- flache TA
  - nur eine strukturelle Abhängigkeit: wenn System-TA Abort durchführt, muss sie es auch

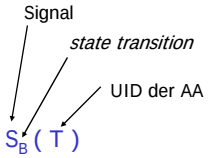


Beschreibung von TA-Modellen (6)

- Beobachtungen
  - Beendete TA kann nicht mehr auf Events reagieren oder ihren Endzustand ändern
  - Flache TA taugen nicht zur Modellierung komplexer Berechnungen; sind vollständig unabhängig von ihrer Umgebung (bis auf Abort durch System-Crash); können sich zu beliebigen Zeitpunkten zurücksetzen oder Commit durchführen
- Regelsprache
  - Regelstruktur
    - <rule identifier> „:"
      [<preconditions>] "→"
      [<rule modifier list>] ","
      [<signal list>] ","
      <state transition>

Beschreibung von TA-Modellen (7)

- Regelsprache (Forts.)
  - Regeln für:
    - BEGIN WORK: S_B (T)
    - ROLLBACK WORK: S_A (T)
    - COMMIT WORK: S_C (T)



- Regelsemantik
  - Regel, die einem Signal zugeordnet ist, wird aktiviert, wenn entsprechendes Event auftritt; sie wird aber erst ausgeführt, wenn <preconditions> erfüllt ist
  - <preconditions> ist einfacher Prädikatsausdruck; oft Prädikate über den Zustand anderer TA , z. B. A(T) oder C(T)

Beschreibung von TA-Modellen (8)

- Regelsemantik (Forts.)
  - <rule identifier> spezifiziert, zu welchem Signaleingang der Pfeil (in der graphischen Darstellung) zeigt; <preconditions> unterscheiden, woher das Signal kommt
  - <signal list> beschreibt, welche Signale bei der <state transition> generiert werden: sie sind Identifikatoren von Regeln, die durch das Signal aktiviert werden (in der graphischen Notation: Endpunkt eines Pfeils)
  - <rule modifier list> fügt zusätzliche Signale in (andere!) Regeln ein oder löscht sie (entsprechen Pfeilen in der graphischen Notation; delete blockiert eine Regel mit allen ihren Signalen)

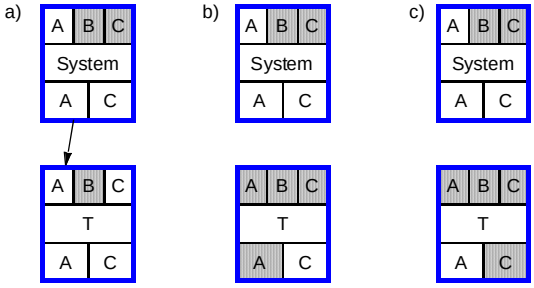
    <rule modifier> ::=
    „+“ „(„ <rule id> „|“ <signal> „)“ |
    „-“ „(„ <rule id> „|“ <signal> „)“ |
    „delete(„ <rule id> „)“

Beschreibung von TA-Modellen (9)

- Regelausführung
  - wenn Event auftritt, identifizierte Regel überprüfen
  - <preconditions> nicht erfüllt: Regel nur markieren, um anzuzeigen, dass Event aufgetreten
  - andernfalls rechte Seite der Regel ausführen (erzeugt i. allg. weitere Signale)
  - alle Aktionen, die von einem Event ausgelöst werden, atomar ausführen; d.h., die Kette abhängiger Events vollständig abwickeln, bevor unabhängiger (von außen kommender) Event betrachtet wird
  - wenn AA ihren Endzustand erreicht hat: alle Regeln löschen, alle ihre belegten Signaleingänge entfernen

Beschreibung von TA-Modellen (10)

- Noch einmal flache TA
  - Beschriebene Zustände
    a) TA T ist aktiv;
    b) TA T hat Commit ausgeführt; Endzustand C
    c) TA T wurde zurückgesetzt; Endzustand A

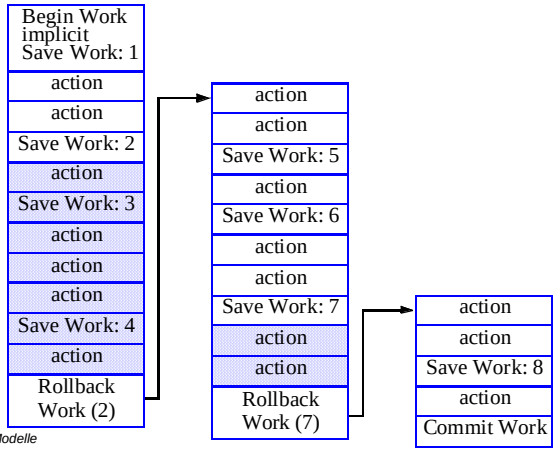


Beschreibung von TA-Modellen (11)

- Noch einmal flache TA (Forts.)
  - Regeln
    - $S_B(T) : \rightarrow$ (+(S_A(system) | $S_A(T)$), delete($S_B(T)$)), , BEGIN WORK
    - $S_C(T) : \rightarrow$ (delete($S_A(T)$), delete($S_C(T)$)), , COMMIT WORK
    - $S_A(T) : \rightarrow$ (delete($S_A(T)$), delete($S_C(T)$)), , ROLLBACK WORK
  - Achtung:
    Im folgenden werden der Übersichtlichkeit halber die delete-Klauseln weggelassen, wenn die Situation offensichtlich ist

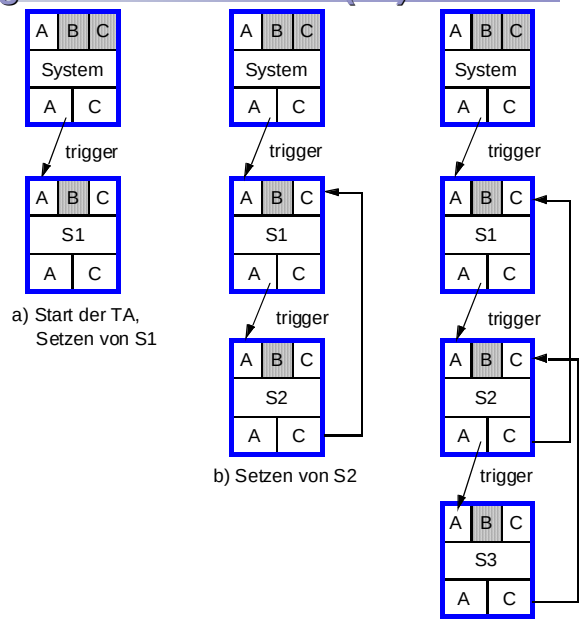
Beschreibung von TA-Modellen (12)

- Flache TA mit Sicherungspunkten
  - Sicherungspunkte innerhalb einer TA
    - werden explizit durch das Anwendungsprogramm gesetzt
    - modifiziertes Rollback benennt einen Sicherungspunkt



Beschreibung von TA-Modellen (13)

- Flache TA mit Sicherungs-punkten (Forts.)
  - Grafische Darstellung



Beschreibung von TA-Modellen (14)

- Flache TA mit Sicherungspunkten (Forts.)
  - Modell
    - Folge von AA
    - verknüpft durch eine Folge von Commits (von der momentanen Position zurück zum TA-Beginn) und
    - verknüpft durch eine Folge von Aborts, ausgehend von der System-TA als Folge eines Crashs
  - Regelmenge
    - S_1 = UID der ersten AA, die ggf. Sicherungspunkt S_1 erzeugt
    - Ziel des Rollback: R (Parameter), spezifiziert als die UID der ältesten AA, bis zu deren Sicherungspunkt zurückzusetzen ist (wenn $R < S_1$, wird die gesamte TA zurückgesetzt)

Beschreibung von TA-Modellen (15)

- Flache TA mit Sicherungspunkten (Forts.)
  - Regelmenge (Forts.)
    - $S_B(S_1)$: $\rightarrow +(S_A(\text{system}) \mid S_A(S_1)),$, BEGIN WORK
    - $S_A(R) : (R < S_1)$ $\rightarrow$, , ROLLBACK WORK
    - $S_C(S_1)$: $\rightarrow$, , COMMIT WORK
    - $S_S(S_1)$: $\rightarrow +(S_A(S_1) \mid S_A(S_2)), S_B(S_2),$

    Regelmenge für AA S_n, die ggf. Sicherungspunkt S_n erzeugt
    - $S_B(S_n)$: $\rightarrow$, , BEGIN WORK
    - $S_A(R): (R < S_n)$ $\rightarrow$, $S_A(S_{n-1})$, ROLLBACK WORK
    - $S_C(S_n)$: $\rightarrow$, $S_C(S_{n-1})$, COMMIT WORK
    - $S_S(S_n)$: $\rightarrow +(S_A(S_n) \mid S_A(S_{n+1})), S_B(S_{n+1}),$

Beschränkungen flacher TA (1)

- Beschränkungen
  - auf kurze Transaktionen zugeschnitten, Probleme mit "langlebigen" Aktivitäten
  - Alles-oder-Nichts-Eigenschaft oft inakzeptabel: hoher Arbeitsverlust
    - keine Binnen-Kontrollstruktur
    - fehlende Kapselung oder Zerlegbarkeit in Teilabläufen
    - keine abgestufte Kontrolle für Synchronisation und Recovery
  - Isolation
    - Leistungsprobleme durch "lange" Sperren
    - fehlende Unterstützung zur Kooperation
  - keine Unterstützung zur Parallelisierung
  - fehlende Benutzerkontrolle

Beschränkungen flacher TA (2)

- Anwendungsbeispiele
  - lange Batch-Vorgänge
    - Beispiel: Zinsberechnung
    - 'Alles-oder-Nichts' führt zu hohem Verlust an Arbeit
    - denkbar: Zerlegung in viele unabhängige Transaktionen – dann jedoch manuelle Recovery nach Systemfehler
  - Mehrschritt-Transaktionen, langlebige Aktivitäten
    - Beispiel: mehrere Reservierungen pro Transaktion
    - lange Sperrdauer (Isolation) führt zu katastrophalem Leistungsverhalten (Sperrkonflikte, Deadlocks)
    - Rücksetzen der gesamten Aktivität im Fehlerfall i.allg. nicht akzeptabel

Beschränkungen flacher TA (3)

- Anwendungsbeispiele
  - Entwurfsvorgänge (CAD, CASE, ...)
    - lange Dauer (Wochen/Monate)
    - kontrollierte Kooperation zwischen mehreren Entwerfern (vor Commit)
    - Einsatz von Versionen
  - Aktive DBS
    - DBS reagiert eigenständig auf bestimmte Ereignisse
    - Spezifikation durch ECA-Regeln
  - Realzeit-Anwendungen
    - zeitbezogene Konsistenzforderungen (Deadlines)
    - häufige irreversible Interaktionen mit der Außenwelt

Beschränkungen flacher TA (4)

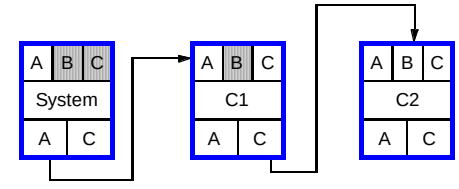
- Anwendungsbeispiele
  - Föderative DBS / Multi-DBS
    - Unterstützung lokaler Knotenautonomie
    - unterschiedliche Synchronisations- und Recovery-Protokolle

Gekettete TA (1)

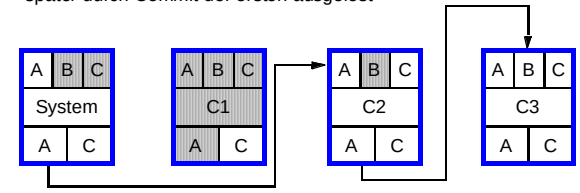
- Variation der Anwendung von Sicherungspunkten
- Modell
  - Folge von atomaren Aktionen (AA), die sequentiell ausgeführt werden
  - Teil der Commit-Verarbeitung: Signal generieren, das BEGIN WORK bei der nächsten AA auslöst
  - Zustandsübergang ist atomar: Chain Work (Commit + Begin)
- Regelmenge
  - $S_B(C_n) : \rightarrow +(S_A(system) \mid S_A(C_n)), , BEGIN\ WORK$
  - $S_A(C_n) : \rightarrow , , ROLLBACK\ WORK$
  - $S_C(S_n) : \rightarrow , S_B(C_{n+1}), COMMIT\ WORK$

Gekettete TA (2)

- Grafische Darstellung



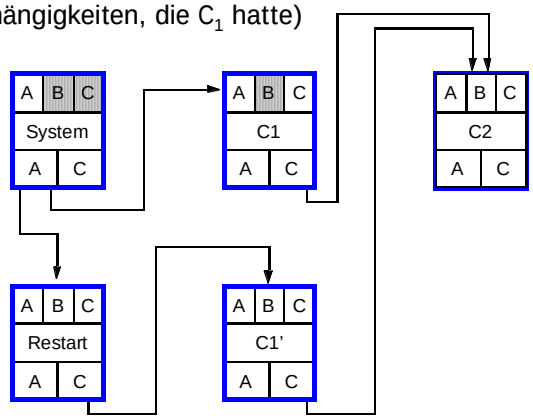
a) Erste TA der Kette wurde gestartet, Start der zweiten TA später durch Commit der ersten ausgelöst



b) Erste TA der Kette hat Commit ausgeführt; zweite TA nun strukturell abhängig von „System"

Gekettete TA (3)

- Gekettete Transaktionen vs. Sicherungspunkte
  - Einführung einer Restart-Aktion
  - Restart wird aktiviert als Folge des Systemausfalls; übernimmt die Rolle von „System" und startet C_1' (mit denselben Abhängigkeiten, die C_1 hatte)



Gekettete TA (4)

- Gekettete Transaktionen vs. Sicherungspunkte
  - Vergleich
    - *Ablaufstruktur:* Gekettete TA erlauben wie Sicherungspunkte Substruktur, die langer Aktivität aufgeprägt werden kann (DB-Kontext bleibt erhalten)
    - *Commit vs. Sicherungspunkt:* Zurücksetzen nur möglich zum letzten SP (= Commit) – vorher: zu beliebigen SP
    - *Sperrbehandlung:* Commit erlaubt Freigabe der Sperren, die später nicht mehr benötigt werden
    - *Verlust von Arbeit:* Sicherungspunkte erlauben flexibleres Zurücksetzen nur, solange System normal arbeitet
    - *Restart-Behandlung:* Bei geketteten TA wird Zustand des jüngsten Commit wiederhergestellt (Problem der Wiederherstellung des AP-Zustandes wie bei SP).

Geschachtelte TA (1)

- Ziele
  - Zerlegung eines Vorgangs (unit of work) in Teilaufgaben: Verteilung und Bearbeitung in einem Rechensystem
    - dynamische Zerlegung einer TA in eine Hierarchie von Sub-TA
    - Bewahrung der ACID-Eigenschaften für die (äußere) TA
    - Gewährleistung von Ununterbrechbarkeit und Isolation für jede Sub-TA
- Vorteile
  - Parallelverarbeitung innerhalb einer Transaktion
    - Ausnutzung anwendungsspezifischer Parallelität
    - Abbildung auf mehrere Prozessoren
  - feinere Recovery-Kontrolle innerhalb einer Transaktion
    - Rücksetzen einer Sub-TA betrifft nur sie und ihre Kinder
    - weitere Verfeinerung durch Sicherungspunkt-Konzept möglich

Geschachtelte TA (2)

- Vorteile (Forts.)
  - explizite Kontrollstruktur
    - einfachere Programmierung paralleler Abläufe
    - 'Alles oder Nichts'-Eigenschaft von Sub-TA reduziert Komplexität
  - Modularität des Gesamtsystems
    - einfache und sichere Zerlegung eines TA-Programmes in Sub-TA
    - unabhängiger Entwurf / Implementierung von Moduln
    - Unterstützung von Kapselung und Fehlerisolation
  - verteilte Systemimplementierung
    - Einsatz verteilter Algorithmen
    - Erhöhung von Verfügbarkeit und Leistung
- Geschachtelte Transaktionen sind eine *Generalisierung* von Sicherungspunkten

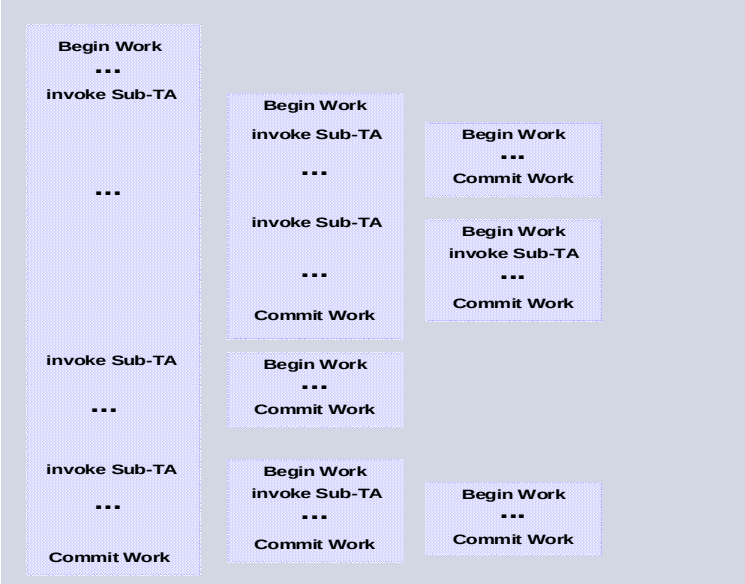
Geschachtelte TA (3)

- Genauere Definition
  - *A nested transaction is a tree of transactions, the sub-trees of which are either nested or flat transactions.*
  - *Transactions at the leaf level are flat transactions. The distance from the root to the leaves can be different for different parts of the tree.*
  - *The transaction at the root of the tree is called top-level transaction, the others are called sub-transactions. A transaction's predecessor in the tree is called parent; a sub-transaction at the next lower level is also called a child.*
  - *A sub-transaction can either commit or rollback; its commit will not take effect, though, unless the parent transaction commits. So, by induction, any sub-transaction can finally commit only if the root transaction commits.*
  - *The rollback of a transaction anywhere in the tree causes all its sub-transactions to roll back. This combined with the previous point is the reason why sub-transactions have only ACI, but not D.*

Geschachtelte TA (4)

- Bemerkung
  - Bemerkung: Die Einhaltung von C lässt sich an die Erzeuger-TA delegieren (größere Flexibilität bei der TA-Zerlegung)
- Dynamisches Verhalten
  - *Commit-Regel:* (lokales) Commit einer Sub-TA macht ihre Ergebnisse nur der Erzeuger-TA zugänglich; endgültiges Commit einer Sub-TA dann und nur dann, wenn für alle Vorfahren bis zur TL-TA das endgültige Commit erfolgreich
  - *Rücksetz-Regel:* wird (Sub-) TA auf irgendeiner Schachtelungsebene zurückgesetzt, werden alle ihre Sub-TA, unabhängig von ihrem lokalen Commit-Status ebenso zurückgesetzt (rekursiv anwenden)
  - *Sichtbarkeits-Regel:* alle Änderungen einer Sub-TA werden bei ihrem Commit für ihre Erzeuger-TA sichtbar; alle Objekte, die Erzeuger-TA hält, können den Sub-TAs zugänglich gemacht werden; Änderungen einer Sub-TA sind für Geschwister-TA nicht sichtbar.

Geschachtelte TA (6)

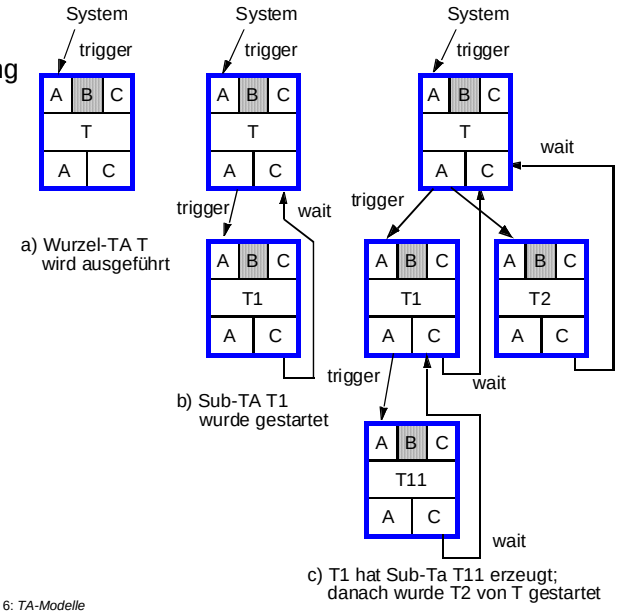


Geschachtelte TA (5)

- Geschachtelte TA sind das Äquivalent der Modularisierung auf der Ebene des Kontrollflusses
  - Jede Sub-TA ist eingebettet in die SoC der Erzeuger-TA
  - Alle ACID-Eigenschaften gelten nur für die TL-TA
- Grafische Darstellung (vgl. nachfolgende Folie)
  - Abhängigkeiten in einer geschachtelten TA sind rein strukturell
  - Abort-Signale werden von oben nach unten durchgereicht
  - Übergang in den Commit-Zustand hängt davon ab, ob Erzeuger-TA ihn schon vollzogen hat

Geschachtelte TA (7)

- Grafische Darstellung (Forts.)



Geschachtelte TA (7)

- Regelmenge
  - für Sub-TA T_{kn} mit Erzeuger-TA T_k
    - $S_B(T_{kn}) : \rightarrow +(S_A(T_k) \mid S_A(T_{kn}))$, , BEGIN WORK
    - $S_A(T_{kn}) : \rightarrow$, , ROLLBACK WORK
    - $S_C(T_{kn}) : C(T_k) \rightarrow$, , COMMIT WORK
  - neu
    - Bedingung für Commit
    - in der graphischen Darstellung Pfeil zurück von Commit-Zustand einer Sub-TA zum Commit-Zustand ihrer Erzeuger-TA (mit „wait“ markiert)

Geschachtelte TA (8)

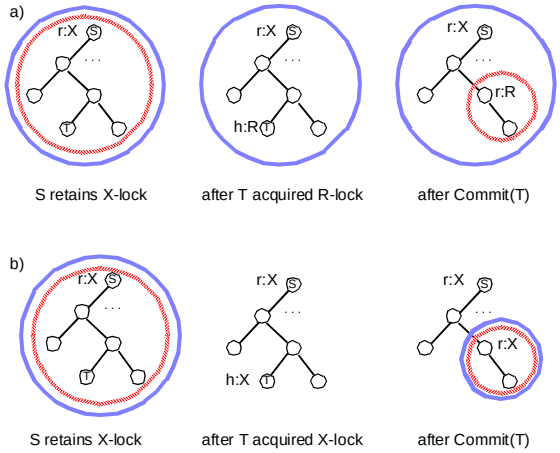
- Vertiefung: Sperren bei flachen Transaktionen
  - Erwerb gemäß Kompatibilitätsmatrix
  - Halten von Sperren: h:(O_1, X), h:(O_2, R)
  - Freigabe bei Commit
- Regeln zum Sperren bei geschachtelten Transaktionen
  - R1: *Transaction T may acquire a lock in X-mode if no other transaction holds the lock in X- or R-mode, and all transactions that retain the lock in X- or R-mode are ancestors of T.*
  - R2: *Transaction T may acquire a lock in R-mode if no other transaction holds the lock in X-mode, and all transactions that retain the lock in X-mode are ancestors of T.*

Geschachtelte TA (9)

- Regeln zum Sperren bei geschachtelten Transaktionen (Forts.)
  - R3: *When a sub-transaction T commits, the parent of T inherits T's locks (held and retained). After that, the parent retains the locks in the same mode (X or R) in which T held or retained the locks previously.*
    - *Note, the inheritance mechanism may cause a transaction to (conceptually) retain several locks on the same object. Of course, the number of locks retained by a transaction should be limited to one by only retaining the most restrictive lock.*
  - R4: *When a transaction aborts, it releases all locks it holds or retains. If any of its superiors holds or retains any of these locks they continue to do so.*
  - Notation
    - X-lock hold on O_1: h:(O_1, X), R-lock retained on O_2: r:(O_2, R)

Geschachtelte TA (10)

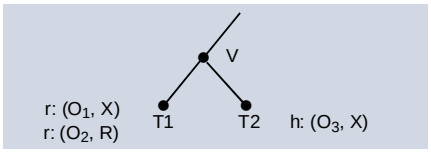
- Vererbung von Sperren *(upward inheritance of locks)*
  - Veränderung der R- und X-Sphären



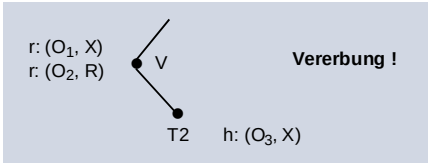
Härder, T., Rothermel, K.: Concurrency Control Issues in Nested Transactions, in: VLDB-Journal 2:1, 1993, pp. 39-74.

Geschachtelte TA (11)

- Vererbung von Sperren *(upward inheritance of locks)*
  - Realisierung
    - Erzeuger-TA erbt Log-Information und Sperren von Sub-TA
    - 'Geerbte Daten' werden verwaltet von einem 'Buchhalter' oder 'TA-Manager'
- Beispiel: Sperren in geschachtelten TA
  1. Ausgangssituation



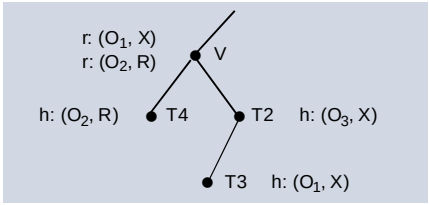
  2. Commit (T1)



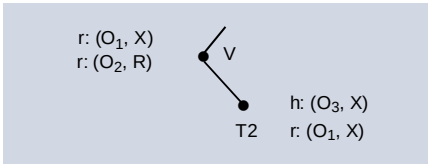
Geschachtelte TA (12)

- Beispiel: Sperren in geschachtelten TA (Forts.)

3. Alle im Kontrollbereich von V können Sperren auf O_1 und O_2 erwerben



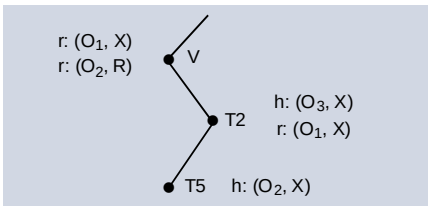
4. T3 und T4 führen Commit durch



Geschachtelte TA (13)

- Beispiel: Sperren in geschachtelten TA (Forts.)

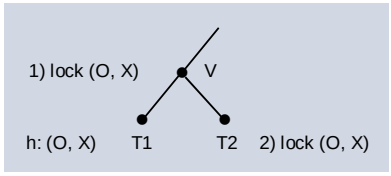
5. T5 fordert (O_2, X) an (keine weitere R-Sperre)



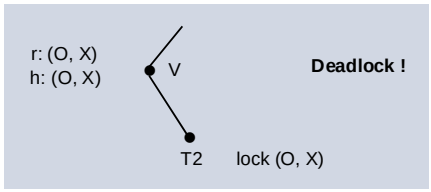
6. V und T5 fordern (O_1, X) an; V muss warten

Geschachtelte TA (14)

- Zusätzliche Deadlock-Probleme

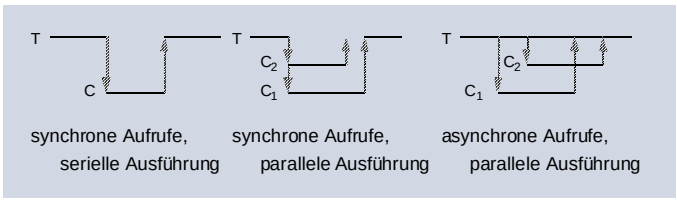


  - Anforderungsreihenfolge wird bei Commit (T1) beibehalten



Geschachtelte TA (15)

- Freiheitsgrade



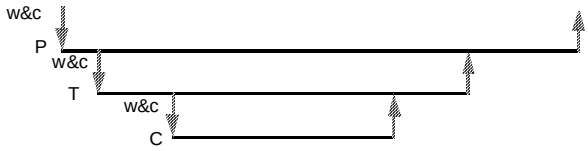
- Repräsentation/Ausführung von (Sub-) TA
  - in einem oder mehreren Prozessen
  - lokale oder verteilte Anordnung
- Client-Server-Beziehung zwischen TA
  - synchroner Aufruf: Client ist blockiert, bis Antwort eintrifft
  - asynchroner Aufruf: Client und Server können parallel arbeiten
  - parallele Initiierung mehrerer (synchroner) Aufrufe (PARBEGIN ... PAREND)

Geschachtelte TA (16)

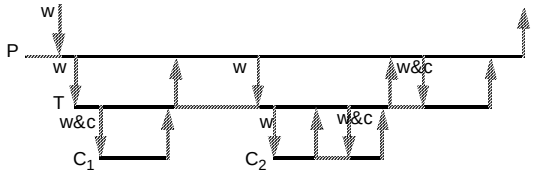
- Schnittstelle zwischen TA
  - "Singe Call"-Schnittstelle:
    TA erzeugt Sub-TA (und diese ggf. rekursiv weitere);
    Antwort impliziert Ende der Sub-TA
  - Konversations-Schnittstelle:
    TA erzeugt Sub-TA; nach einer Antwort bleibt der Kontext der Sub-TA erhalten; sie kann weitere Anforderungen bearbeiten, bis explizit EOT der Sub-TA ausgeführt wird.
- Kooperation (vgl. Illustration nachfolgende Folie)
  - Aufrufprimitive
    work (w), work & commit (w&c), commit (c), accept, abort, done
  - Forderung: "Hierarchical Containment" bei Konversationsschnittstelle
    - Vereinfachung der Schachtelungsstruktur
    - Begrenzung des Domino-Effekts usw.

Geschachtelte TA (17)

- Kooperation



Benutzung von Single-Call-Schnittstelle



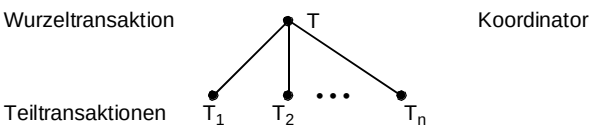
Benutzung von Konversations-Schnittstellen

Geschachtelte TA (18)

- 4 Arten von “Intra-Transaction”-Parallelität
  - Serielle Ausführung von TA: synchrone Aktivierung (z.B. “remote procedure call”)
  - Nur Parallelität zwischen Kind-TA (sibling parallelism): parallele Aktivierung mehrerer synchroner Aktionen (z.B. ARGUS)
  - Vater/Kind-Parallelität: Nur parallele TA-Ausführung in hierarchischem Pfad
  - Uneingeschränkte Parallelität: asynchrone TA-Aktivierungen auf allen Ebenen (z.B. CLOUDS und LOCUS)

Verteilte TA (1)

- Verteilte Transaktion ist typischerweise flache TA,
  - die in einer verteilten Umgebung abläuft und deshalb mehrere Knoten im Netz aufsucht
  - deren Verteilung von den Daten abhängt
  - die in „Scheiben“ derselben Top-Level-TA aufgeteilt ist
- Struktur einer geschachtelten TA wird dagegen von funktionaler Zerlegung bestimmt!
- Kontrollstruktur in einer verteilten Umgebung

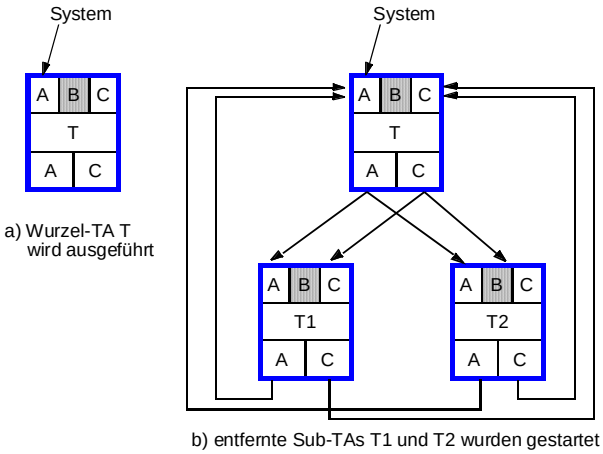


Verteilte TA (2)

- Behandlung von
  - Isolation (Sperren),
  - Konsistenzsicherung,
  - Rücksetzen, Commit?
- Kopplung zwischen Sub-TA und ihrer übergeordneten TA in diesem Modell viel stärker als bei geschachtelten TA
- Regelmenge (für den Fall einer Sub-TA T1)
  - $S_B(T) : \rightarrow +(S_A(system) \mid S_A(T)), , BEGIN\ WORK$
  - $S_A(T) : \rightarrow , , ROLLBACK\ WORK$
  - $S_C(T) : \rightarrow , , COMMIT\ WORK$
  - $S_B(T1) : \rightarrow (+(S_A(T) \mid S_A(T1)), +(S_C(T) \mid S_C(T1))), , BEGIN\ WORK$
  - $S_A(T1) : \rightarrow , S_A(T), ROLLBACK\ WORK$
  - $S_C(T1) : \rightarrow , S_C(T), COMMIT\ WORK$

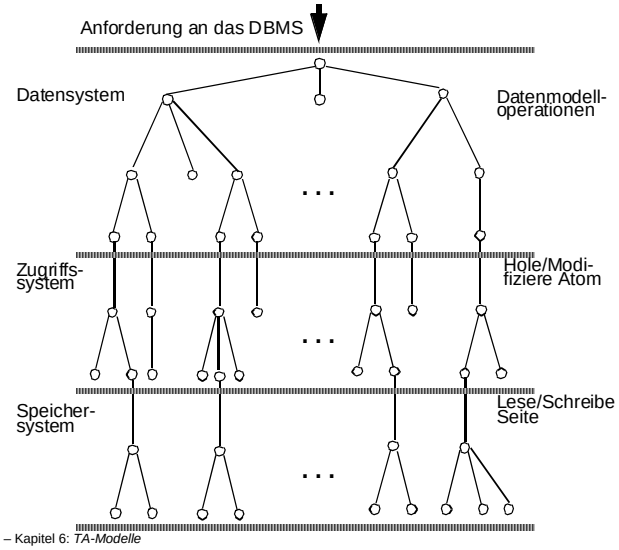
Verteilte TA (3)

- Grafische Darstellung



Mehrebenen-TA (1)

- Motivation: Dynamische Aufrufhierarchie in einem DBS (Bsp.)



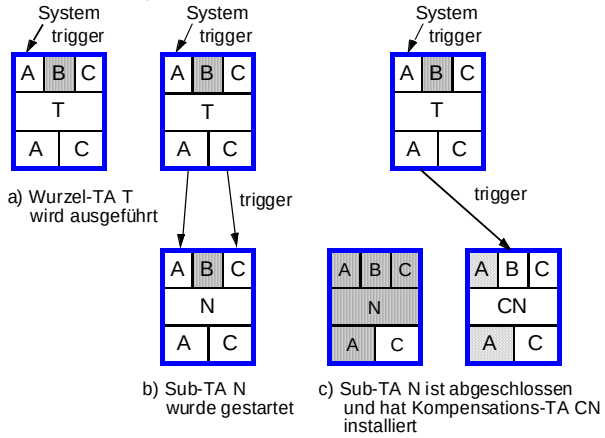
Mehrebenen-TA (2)

- Motivation (Forts.)
  - Hoher Leistungsbedarf bei komplexen Anforderungen, z. B bei Checkout/Checkin- oder bei interaktiver Anfrage-Verarbeitung
    - Reduktion der Antwortzeit, Erhöhung der Verfügbarkeit
    - Parallelisierung und Verteilung von Operationen
    - feinere Recovery-Granulate

    ⇨ Bildung von Subtransaktionen im DBS-Kern
- Verallgemeinerung geschachtelter TA: offene Schachtelung
  - Sub-TA dürfen vor Ende der Erzeuger-TA Commit ausführen ('pre-commit'), aber nicht umgekehrt!
  - danach einseitiges Zurücksetzen (Rollback) nicht mehr möglich
  - aber: Kompensation
    - Wirkung der Sub-TA rückgängig machen, wenn Erzeuger scheitert
    - kompensierende TA selbst wieder geschachtelte TA oder Mehrebenen-TA

Mehrebenen-TA (3)

- Grafische Darstellung



  - Ein Abort darf nicht fehlschlagen! Kompensierende TA CN muss Commit durchführen; Wirkung einer Blockierung des A-Eingangs und des A-Zustands wird durch Restart-TA CN' realisiert;

Mehrebenen-TA (4)

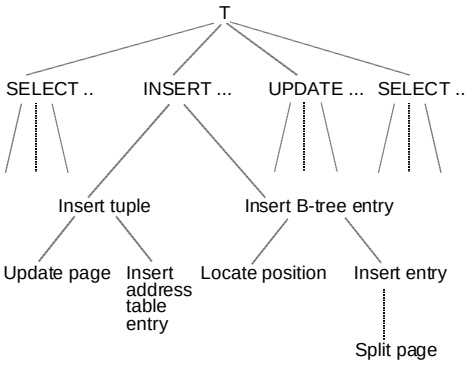
- Kompensations-TA
  - installieren, wenn Sub-TA Commit ausführt
  - aufbewahren, solange Erzeuger-TA noch läuft
  - nach Abschluss der Erzeuger-TA wegwerfen
  - wichtig: Kompensations-TA muss Commit erreichen!
- Regeln für Sub-TA N
  - Regeln für Wurzel-TA T identisch zu denen für flache TA
  - $S_B(N) : \rightarrow (+(S_A(T) \mid S_A(N)), +(S_C(T) \mid S_C(N))), , \text{BEGIN WORK}$
  - $S_A(N) : \rightarrow , , \text{ROLLBACK WORK}$
  - $S_C(N) : \rightarrow +(S_A(T) \mid S_B(CN)), , \text{COMMIT WORK}$

Mehrebenen-TA (5)

- Regeln für Kompensations-TA CN
  - Vorsorge für den Fehlerfall während des CN-Ablaufs
  - CN im Falle des Scheiterns neu starten als CN'
    (vgl. gekettete TA)
    - $S_B(CN) : \rightarrow +(S_C(restart) \mid S_B(CN'))$, , BEGIN WORK
    - $S_A(CN) : \rightarrow$, $S_B(CN')$, ROLLBACK WORK
    - $S_C(CN) : \rightarrow delete(S_B(CN'))$, , COMMIT WORK
  - Beachte
    - CN' hat denselben Code, erhält dieselben Daten, hat aber einen anderen Namen; eine AA kann nur einmal ausgeführt werden, d. h. unter anderem, dass mit ihr kein Restart ausgeführt werden kann;

Mehrebenen-TA (6)

- Nutzung: Strukturierung der Operationshierarchie im Schichtenmodell

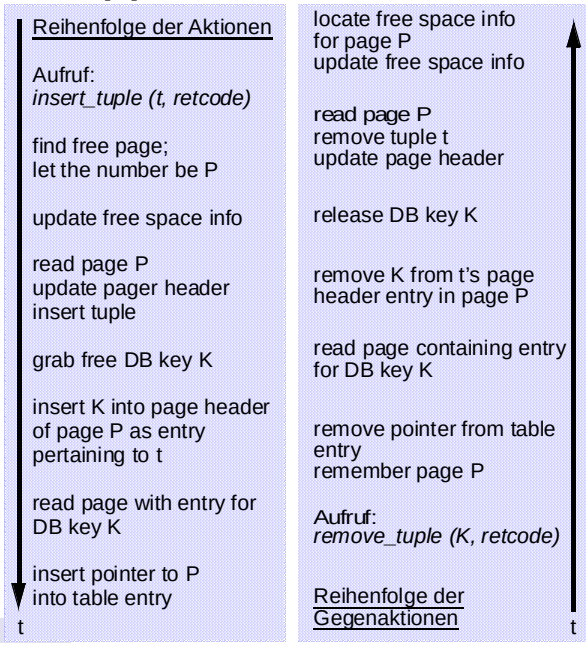


Mehrebenen-TA (7)

- Nutzung: Strukturierung der Operationshierarchie im Schichtenmodell (Forts.)
  - SQL-Anweisungen können als Sub-TA aufgefasst werden
  - ebenso die internen Modulaufrufe
  - bei geschachtelten TA: Sperren auf Seiten, Adresstabellen, B-Baum-Seiten (!) bleiben erhalten (bei der Erzeuger-TA)
  - bei Mehrebenen-TA: Sperren werden freigegeben! z.B. Seite für andere Sub-TA zugänglich; Kompensation: Tupel wieder aus Seite löschen
  - Voraussetzung dafür ist, dass eine Gegenaktion zur Verfügung steht, welche die ursprüngliche Operation kompensieren kann
  - Zu Darstellung auf nachfolgender Folie
    - Entfernung des Tupels funktioniert auch, wenn Seite inzwischen von anderen TA geändert wurde

Mehrebenen-TA (7)

- Nutzung (Forts.)



Mehrebenen-TA (8)

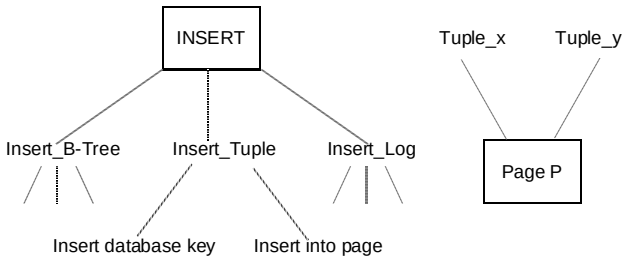
- Strukturelle Voraussetzungen für den Einsatz von Mehrebenen-TA
  - Abstraktionshierarchie: Gesamtes System besteht aus strikter Hierarchie von Objekten mit ihren Operationen
  - Schichtenweise Abstraktion: Objekte der Schicht n werden vollständig implementiert unter Verwendung von Operationen der Schicht n-1
  - Disziplin: Schicht n darf nur auf Objekte von Schicht n-1 zugreifen

Mehrebenen-TA (9)

- Wirkungsprinzip der Mehrebenen-TA
  - beruht nicht auf einer Enthaltenseinshierarchie,
  - sondern auf Abbildungshierarchie der Objekte
    - Jede Ebene übt Commit-Kontrolle über die auf ihr anfallenden Objektänderungen aus
    - Objekthierarchie schützt implizit alle Objekte auf niedrigeren Ebenen, die zur Implementierung von Objekten auf höherer Ebene benötigt werden
    - Bei Objekthierarchie Tupel - Seite - Block erlaubt eine Tupelsperre die vorzeitige Freigabe der Seitensperre, die zum Einfügen benötigt wird

Mehrebenen-TA (10)

- Wirkungsprinzip der Mehrebenen-TA (Forts.)

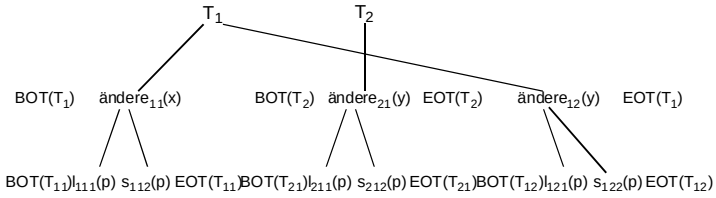


Mehrebenen-TA (11)

- Schachtelung der TA
  - lässt sich auf beliebige Schichten (beliebige Operationen) verallgemeinern
  - theoretisch fundierter Ansatz
- Transaktionsverwaltung in jeder Schicht
  - Ausnutzung von Anwendungssemantik zur Synchronisation möglich
  - Wahl unterschiedlicher Synchronisationstechniken pro Schicht möglich
  - potentiell hoher Aufwand zur TA-Verwaltung, insbesondere für Logging und Recovery (Protokollierung in mehreren Schichten)

Mehrebenen-TA (12)

- Beispiel: serialisierbarer Ablaufplan mit 2 Ebenen



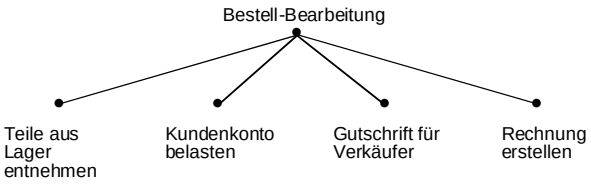
- Wesentliche Eigenschaften
  - reduzierte Konfliktgefahr zwischen TA auf niedrigeren Ebenen unter Wahrung von Serialisierbarkeit
  - vorzeitiges Commit (Freigabe von Änderungen/Sperren) von Sub-TA
  - aber: „Schutzschirm" auf höherer Ebene bleibt erhalten (z.B. striktes 2-Phasen-Sperrprotokoll für TL-TA)
  - für Gesamt-TA gelten weiterhin ACID-Eigenschaften

Offen geschachtelte TA (1)

- ACID für langlebige TA?
  - sehr lange Sperren
  - Sperren vieler Objekte → Erhöhung der Blockierungsrate und Konfliktrate
  - höhere Rücksetzrate (Deadlock-Häufigkeit stark abhängig von TA-Größe)
  - höhere Chance, durch einen Systemfehler zurückgesetzt zu werden
  - Leistungsprobleme
- 'Anarchische' Variante der Mehrebenen-TA
  - keine Restriktionen bzgl. semantischer Beziehung zwischen Sub-TA und Erzeuger-TA
  - insbesondere auch keine Objekthierarchie

Offen geschachtelte TA (2)

- Beispiel



  - Subtransaktionen T_i arbeiten oft auf verschiedenen Datenbeständen

- Prinzipien
  - Sperren werden vor Beendigung der TL-TA freigegeben
  - für jede ändernde Sub-TA T_i wird Kompensations-TA CT_i bereitgestellt
  - im Fehlerfall: Ausführung von CT_i (kein UNDO(T_i))

⇒ keine Serialisierbarkeit

Langlebige TA (1)

- Verarbeitung langer Batch-Anwendungen
- Einsatz von flachen Transaktionen?
  - kontextfreie Verarbeitung: output_msg = f (input_msg)
  - “Exactly once”-Semantik wird eingehalten
  - Kosten im Restart-Fall
  - Sicherungspunkt oder geschachtelte TA helfen bei Crash nicht
- Zerlegung in Mini-Batches
  - Nutzung von Verarbeitungskontexten
  - Hintereinander-Ausführung der TA-Folge unter Beibehaltung des TA-Verarbeitungskontextes
  - output_msg = f (input_msg, context)

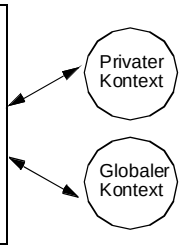
Langlebige TA (2)

- Kontextinformationen
  - *Transaktion:* Cursorpositionen, ...
  - *Programm:* letzte TA, die erfolgreich Commit durchgeführt hat, ...
  - *Terminal:* Liste der Funktionen, die aufgerufen werden können; letztes ausgegebenes Fenster; Liste der Benutzer, die das Terminal benutzen dürfen, ...
  - *Benutzer:* letzter Auftrag, den der Benutzer bearbeitet hat; nächstes zu benutzendes Passwort, ...

Langlebige TA (3)

- Einsatz von Verarbeitungskontexten
  - Ausführung eines kontext-sensitiven TA-Programms beruht auf
    - den Parametern in der Eingabe-Nachricht
    - und Zustandsinformationen als Kontext

```
SimpleProgram()
 { read (input_message);
  BEGIN WORK;
  /* perform computation based on
     input message and context */;
  send (output_message);
 COMMIT WORK;
 };
```



  - Ergebnis
    - ist eine Ausgabenachricht
    - und ein geänderter Kontext

Langlebige TA (3)

- Eigenschaften langlebiger Transaktionen
  - Minimierung der verlorengegangenen Arbeit im Fehlerfall
  - wiederherstellbarer Verarbeitungszustand
  - expliziter Kontrollfluss
  - Einhaltung der ACID-Eigenschaften (für Einzel-TA)

Langlebige TA (4)

- Beispiel: Zinsberechnung

```
ComputeInterest (interest_rate) {

#include <string.h>
#include <sqlca.h>
#define max_account_no 999999

exec sql begin declare section;
  long last_account_done;
  double interest_rate;
  int logsize;
exec sql end declare section;

exec sql define stepsize 1000;

/* Annahme: Kontonr. 1 bis 1000000,
Relation batchcontext enthält id des
letzten Mini-Batches */

logsize = 0;
exec sql select count (*) into :logsize
         from batchcontext;

if (sqlca.sqlcode != 0 | | logsize = = 0)   {
/* batchcontext entweder nicht vorhanden oder
leer → Anfang der Kette */
  exec sql begin work;
  exec sql drop table batchcontext;
  exec sql create table batchcontext
     (last_account_done integer);
  last_account_done = 0;
  exec sql insert into batchcontext
     values (: last_account_done);
  exec sql commit work;
}
```

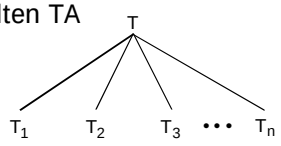
Langlebige TA (4)

- Beispiel: Zinsberechnung (Forts.)

```
else   {
  /* Restart */
  exec sql select last_account_done
    into :last_account_done
    from batchcontext;
}
while (last_account_done < max_account_no)  {
  /* ein Mini-Batch: */
  exec sql begin work;
  exec sql update accounts
    set account_total = account_total
        * (1 + :interest_rate)
    where account_no between
      : last_account_done + 1 and
      : last_account_done + :stepsize;
  exec sql update batchcontext
    set last_account_done =
    last_account_done + :stepsize;
  exec sql commit work;
  last_account_done =
    last_account_done + stepsize;
}
/* letzter Mini-Batch ausgeführt */
exec sql begin work;
exec sql drop table batchcontext;
exec sql commit work;
return;
}
```


Sagas (1)

- Linderung der LLT-Probleme bei speziellen Anwendungen
  - vorzeitige Freigabe von Ressourcen
  - LLT nicht mehr atomar, jedoch keine Preisgabe der DB-Konsistenz
  - Koordinierte Fehlerbehandlung bzw. Rücksetzung erforderlich
- Spezielle Art von zweistufigen geschachtelten TA
  - Saga ≡ LLT, die in eine Sammlung von Subtransaktionen aufgeteilt werden kann



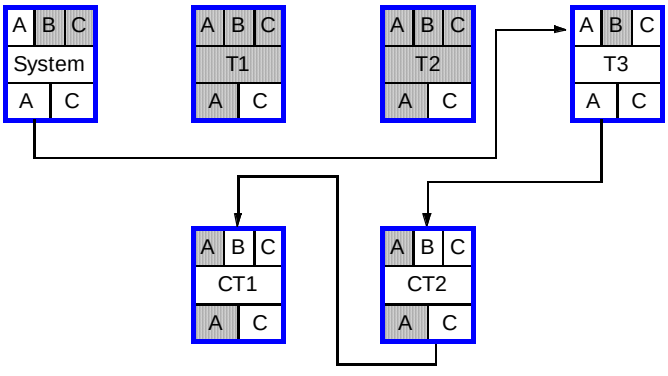
- Aspekte der Ablaufsteuerung und -kontrolle
  - T_i geben alle Ressourcen (z. B. Sperren) frei
  - expliziter Kontrollfluss zwischen den T_i (Sequenz) im AW-Programm
  - Synchronisation verlangt Serialisierbarkeit der T_i, jedoch nicht von T
  - Konfliktbehandlung durch Warten oder Abbruch von T_i oder von T
  - Fehlerbehandlung von T_i durch UNDO(T_i) und Kompensation CT_{i-1}, ...,CT_1

Sagas (2)

- Es muss für Kompensation gesorgt werden!
  - alle T_i gehören zusammen
  - Bereitstellung von Kompensationstransaktionen CT_i für jede T_i
  - keine teilweise Ausführung von T (!)
- Saga: Menge flacher Transaktionen $T_1, T_2, ..., T_n$, die im einfachsten Fall sequentiell verarbeitet wird
- Zusicherung des DBS
  - Das Endergebnis einer Saga ist entweder die Ausführungsfolge $T_1, T_2, ..., T_i, ..., T_{n-1}, T_n$
  - oder bei einem Fehler im Schritt j
    $T_1, T_2, ..., T_j$ (abort), $CT_{j-1}, ..., CT_2, CT_1$
  - DBS garantiert LIFO-Ausführung der Kompensationen im Fehlerfall
- $T_j, CT_j \neq UNDO(T_j)$ im allgemeinen Fall
  - für CT_j müssen Ressourcen wieder angefordert werden
  - Deadlock-Gefahr

Sagas (3)

- Grafische Darstellung (Momentaufnahme mit Planung künftiger TA)



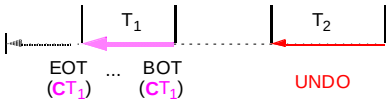
Sagas (4)

- Zusätzliche Kontroll-Operationen:
  BEGIN_SAGA (BS), ABORT_SAGA (AS), END_SAGA (ES)
- Ablauf

a) BS ... BOT(T_1) ... EOT(T_1) ... BOT(T_2) ... EOT(T_2) ... EOT(T_n) ... ES

b) BS ... BOT(T_1) ... EOT(T_1) ... BOT(T_2) ... ABORT

c) BS ... BOT(T_1) ... EOT(T_1) ... BOT(T_2) ... ABORT_SAGA



d) Unterbrechung durch Systemfehler: wie Fall c

Sagas (5)

- Eigenschaften
  - ACID für jede Sub-TA T_i
    (D kann durch Kompensation aufgehoben werden)
  - CD für umfassende TA T
- Verfeinerung
  - Reduktion des Aufwandes beim Scheitern einer Saga oder bei Systemfehler durch Nutzung von Savepoints
    - Sicherung des AP-Zustandes
    - Übergabe der Savepoint-ID an ABORT_SAGA
  - Partielles Rücksetzen möglich
  - Szenario
    - Savepoint nach T_1 und T_3
    - Crash nach T_2 und T_5
  - Ablauf
    - BS, T_1, T_2, CT_2, T_2, T_3, T_4, T_5, CT_5, CT_4, T_4, T_5, T_6, ES

Sagas (6)

- C_i ist Anwendungsprogramm (vordefinierte TA)
  - Speicherung in DB vorteilhaft
  - keine unkontrollierten Programme
  - aktuelle Parameter für CT_i aus DB

⇒ automatische Recovery möglich!

Zusammenfassung (1)

- Transaktionskonzept (ACID) hat sich durchgesetzt
  - gut verstanden
  - leistungsfähige Implementierungen
  - Einsetzbarkeit über den DB-Bereich hinaus (Dateisysteme, Mail-Service, ...)
- Geschlossen geschachtelte Transaktionen
  - Unterstützung von Intra-Transaktionsparallelität
  - feinere Rücksetzeinheiten
  - v.a. in verteilten Systemen wichtig

Zusammenfassung (2)

- Mehreben-Transaktionen
  - Abbildungshierarchie von Objekten
  - Reduzierung der Konfliktgefahr zwischen TA auf niedrigeren Ebenen durch vorzeitiges Commit (Freigabe von Änderungen/Sperren) von Sub-TA
  - aber: „Schutzschirm" auf höherer Ebene
- Offen geschachtelte Transaktionen (z.B. Sagas)
  - Unterstützung langlebiger Transaktionen
  - Reduzierung der Konfliktgefahr durch vorzeitige Sperrfreigabe (erhöhte Inter-Transaktionsparallelität)
  - Recovery durch Kompensation

Ausblick – Entwurfs-TA

- TA-Konzept in Entwurfsanwendungen
  - anwendungsbezogene Definition
  - Nutzung von Semantik bei Synchronisation und Recovery
  - Rücksetzen auf Binnen-Sicherungspunkte
  - Nutzung von Versionen
  - Ggf. Orientierung der Recovery und Synchronisation an AW-Objekten
- Unterstützung der Kooperation
  - zugeschnittene Verarbeitungsmodelle
  - Workstation/Server-Architektur
  - Unterstützung von Gruppenarbeit
  - (System-)Kontollierte Kooperation (Delegation von Teilaufträgen, Austausch vorläufiger Entwurfsdaten, Verhandeln von Entwurfszielen)

Abschließender Vergleich

	geschachtelte TA	Mehrebenen-TA offen geschacht. mit Disziplin	offene geschachtelte TA	Batch-TA (langlebige TA)	Sagas	Entwurfs-TA (Ausblick)
expliziter Kontrollfluß (außerhalb TA)	-	-	-	einfache Sequenz	Verkettung, einfache Sequenz	-
frühzeitige Freigabe von Änderungen	-	nur relativ zur selben Schicht	ja	frühestens am Ende einer Teil-TA	frühestens am Ende einer Teil-TA	im Kooperationsmodus
stabile Ergebnisse nach Systemausfall	-	-	für die TA, die unabhängig Commit gemacht haben	für die schon beendete Teil-TA	-	nur wenn Objektversion in Gruppen-DB eingebracht
Begrenzung der dynamischen Rücksetzung (Rollback)	ja	ja	ja	-	-	-
Korrektheit	Serialisierbarkeit	Serialisierbarkeit	-	Serialisierbarkeit der Einzel-TA	Serialisierbarkeit der Einzel-TA	Kooperation auf vorläufigen Objekten
Kontext-Verwaltung für zusammenhängende Abläufe	-	-	-	teilweise	-	-
explizte Konfliktbehandlung	-	-	-	-	-	Grant-/Return-Protokoll