Stratifizierte Transaktionen (4)

- **Vorteile stratifizierter Transaktionen**
  - im Vergleich zu T: früheres Commit der einzelnen Strata S_i

  ➥ impliziert frühere Freigabe der Sperren und
  damit höhere Nebenläufigkeit

  - Antwortzeit für Benutzer:
    Ausführungszeit des Wurzelstratums (weniger TA!)

  - 2PC-Protokolle für alle Strata können weniger Nachrichten umfassen
    als das 2PC-Protokoll einer globalen TA T

  - Kolokation als mögliches Kriterium für Stratifikation von T

  ➥ lokale 2PC-Nachrichten

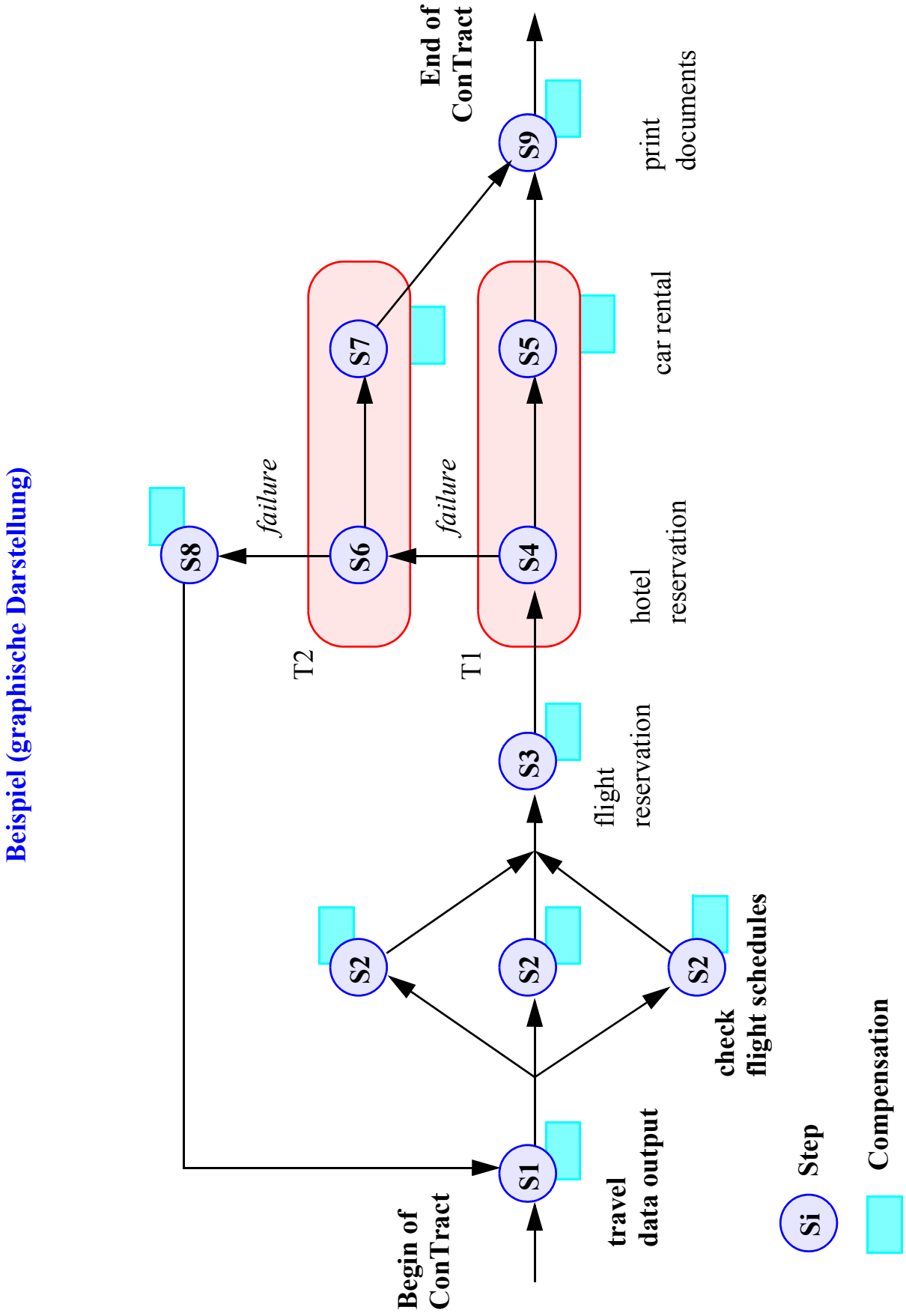
ConTracts (1)

- **ConTracts:[12] Erweiterung des Saga-Ansatzes um**
  - reichere Kontrollstrukturen
    (Sequenz, Verzweigung, Parallelität, Schleife usw.)
  - getrennte Beschreibung von Sub-Transaktionen (**Steps**) und Ablaufkontrolle (**Skript**)
  - Verwaltung eines persistenten **Kontextes** für globale Variablen, Zwischenergebnisse, Bildschirmausgaben usw.
  - Synchronisation zwischen Steps über Invarianten
  - flexiblere Konflikt-/Fehlerbehandlung

  ➥ **einer der ersten innovativen Forschungs-Prototypen**

12. Wächter, H., Reuter, A.: The Contract Model, in Elmagarmid, A.K. (Hrsg.): Transaction Models for Advanced Applications, Morgan Kaufmann, San Mateo, CA, 1992, S. 219-264.

ConTracts (2)


Beispiel (graphische Darstellung)
Begin of ConTract
travel data output
check flight schedules
flight reservation
hotel reservation
car rental
print documents
End of ConTract
failure
failure
T2
T1
S1
S2
S2
S2
S3
S4
S5
S6
S7
S8
S9
Si
Step
Compensation

ConTracts (3)

- **Beispiel (Forts.)**
  - zugehöriges Skript

```
CONTRACT Business_Trip_Reservations

CONTEXT_DECLARATION
    cost_limit, ticket_price:      dollar;
    from, to:                      city;
    date:                          date_type;
    ok:                            boolean;

CONTROL_FLOW_SCRIPT

    S1: Travel_Data_Input (   in_context:     ;
                              out_context: data, from, to, cost_limit );

    PAR_FOREACH ( airline: EXECSQL select airline from ... ENDSQL )
        S2: Check_Flight_Schedule (    in_context:   airline, data, from, to;
                                       out_context: flight_no, ticket_price );
    END_PAR_FOREACH;

    S3: Flight_Reservation (   in_context:   flight, ticket_price; ... );
    S4: Hotel_Reservation (    in_context:   "Cathedral Hill Hotel";
                               out_context: ok, hotel_reservation );

    IF ok THEN

        S5: Car_Rental ( ... "Avis" ... );

    ELSE BEGIN

        S6: Hotel_Reservation ( ... "Holiday Inn" ... );
        IF ok THEN
            S7: Car_Rental ( ... "Hertz" ... );
        ELSE S8 : Cancel_Flight_Reservation_&_Try_Another_One ( ... );
        END

    S9: Print_Documents ( ... );

END_CONTROL_FLOW_SCRIPT

COMPENSATIONS

        C1: Do_Nothing_Step();
        C2: Do_Nothing_Step();
```

ConTracts (4)

- **Beispiel (Forts.)**
  - zugehöriges Skript (Forts.)

```
        C3: Cancel_Flight_Reservation( ... );
        C4: Cancel_Hotel_Reservation( ... );
        C5: Cancel_Car_Reservation( ... );
        C6: Cancel_Hotel_Reservation( ... );
        C7: Cancel_Car_Reservation( ... );
        C8: Do_Nothing_Step();
        C9: Invalidate_Tickets( ... );

    END_COMPENSATIONS

    TRANSACTIONS

        T1 (S4, S5), DEPENDENCY ( T1:abort → begin:T2 );
        T2 (S6, S7), DEPENDENCY( T2:abort → begin:S8 );

    END_TRANSACTIONS

    SYNCHRONIZATION_INVARIANTS_&_CONFLICT_RESOLUTIONS

        S1:   EXIT_INVARIANT (budget > cost_limit);
              POLICY:          check/revalidate;

        S3:   ENTRY_INVARIANT    (budget > cost_limit) AND
                                 (cost_limit > ticket_price));
              CONFLICT_RESOLUTION: S8: Cancel_Reservation ( ...) ;
              EXIT_INVARIANT (budget > cost_limit - ticket_price);
              POLICY:          check/revalidate;

        S4, S6: ENTRY_INVARIANT (hotel_price < budget);
              CONFLICT_RESOLUTION:
                    S10: Call_Manager_To_Increase_Budget ( ... );

        S5, S7: ENTRY_INVARIANT (car_price < budget);
              CONFLICT_RESOLUTION:
                    S10: Call_Manager_To_Increase_Budget ( ... );

    END_SYNCHRONIZATION_INVARIANTS_&_CONFLICT_RESOLUTIONS

    END_CONTRACT Business_Trip_Reservations.
```

ConTracts (5)

- **Programmiermodell**
  - Programmierung von Steps ist unabhängig von der Erstellung von Skripten
  - Beispiel für einen Step (Fragment):

```
STEP Flight_Reservation

DESCRIPTION: Reserve n seats of a flight and pay for them ...

IN      airline:        STRING;
        flight_no:      STRING;
        date:           DATE;
        seats:          INTEGER;
        ticket_price:   DOLLAR;

OUT     status:         INTEGER;

flight_reservation ()

{       char*   flight_no;
        long    date;
        int     seats;
        ...
        EXEC SQL
                UPDATE  Reservations
                SET     seats_taken = seats_taken + :seats
                WHERE   flight = :flight_no AND
                        date = :date ...
        END SQL
        ...

}
```

ConTracts (6)

- **Transaktionsmodell**
  - Steps: ACID (lokal)
  - Atomare Einheiten

```
TRANSACTIONS

    T1 (S4, S5),
    T2 (S6, S7),

END_TRANSACTIONS
```

  - Schachtelung möglich

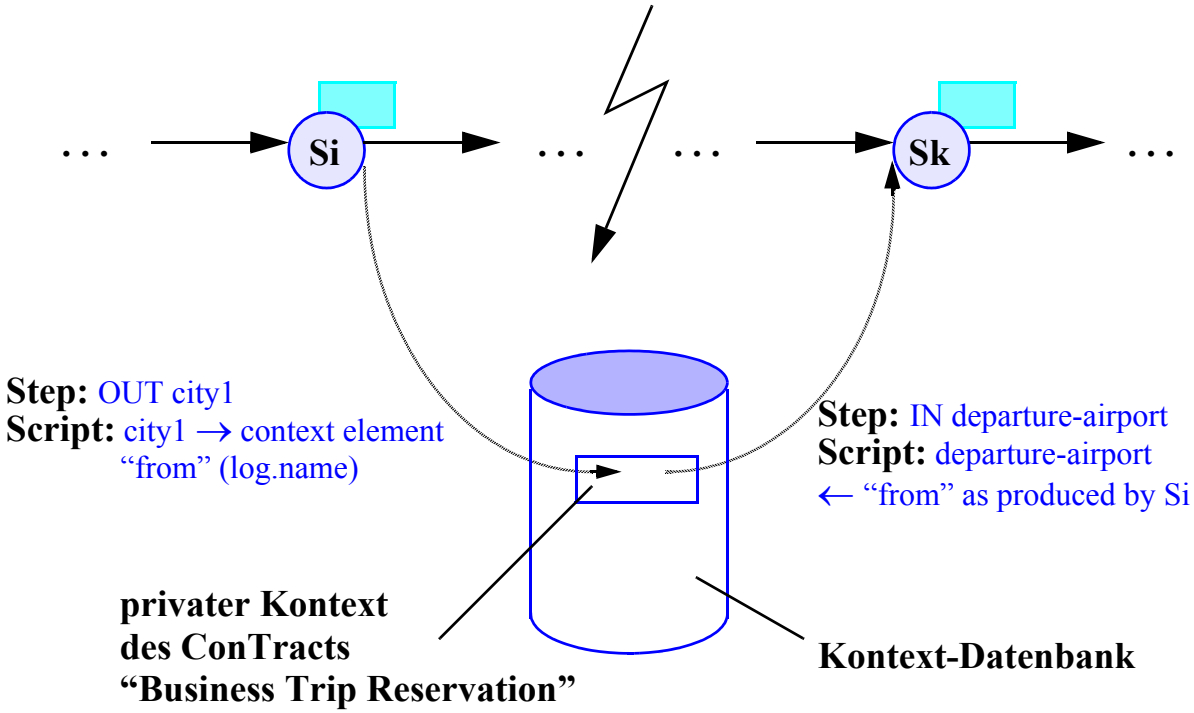
```
T1 (T2, T3 )
```

  - Abhängigkeiten
    - Alternative zu obigem Beispiel

```
T1 (S4, S5),
  DEPENDENCY( T1:abort[1] → begin:T1 );
                              /* erster Abort von T1 */
  DEPENDENCY(T1:abort[2] → begin:S8 );
                              /* zweiter Abort von T1 */
```

ConTracts (7)

- **Forward Recovery und Kontext-Management**
  - *Forward Recovery*: nach einem Fehler soll auf dem jüngsten Step-konsistenten Zustand wiederaufgesetzt werden und ‘nach vorne’ weiterverfahren werden
  - Forward Recovery erfordert persistentes *Kontext-Management*



  - Attribute von Kontextelementen:
    logischer Name,
    ConTract-Identifikator,
    Step-Identifikator,
    Zeitpunkt der Erzeugung,
    *Version*snummer
    (bei mehreren Aktivierungen desselben Steps),
    Zähler (für parallele Aktivierungen);

ConTracts (8)

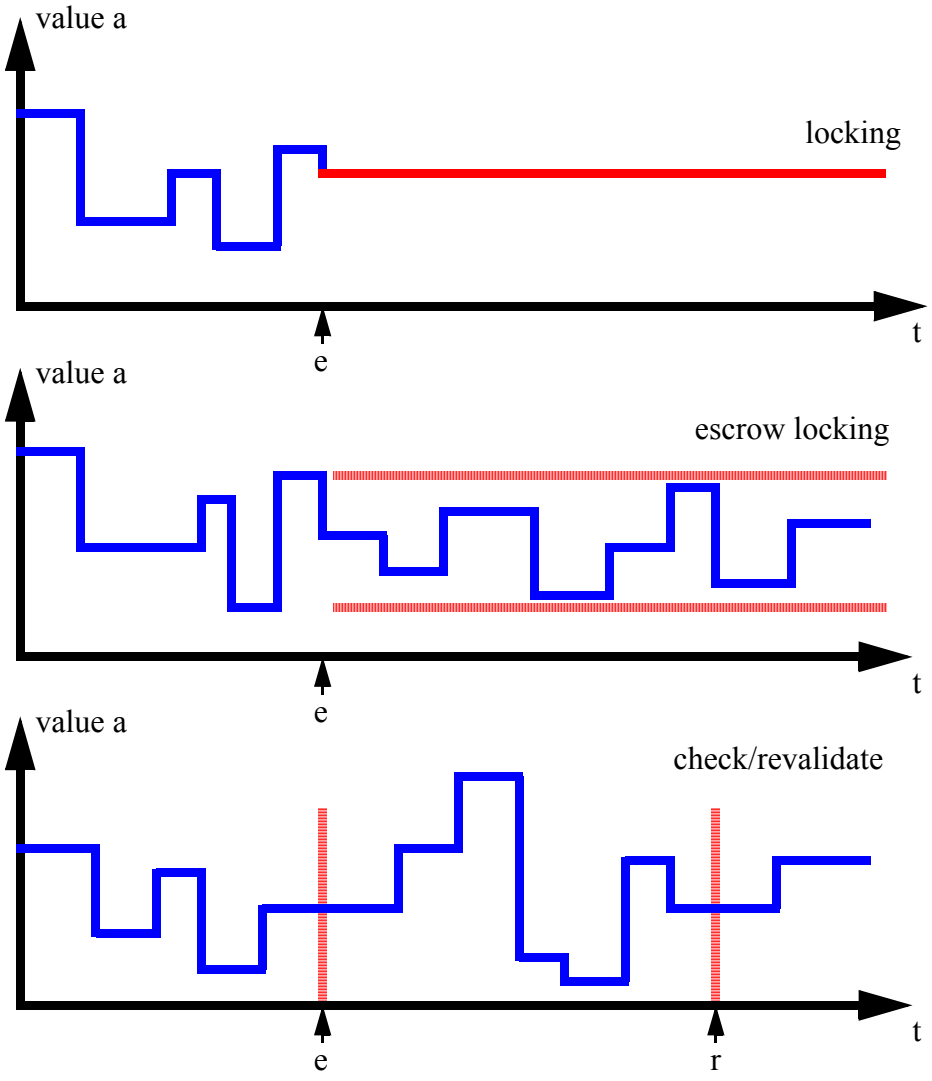
- **Kompensation**
  - Kompensation eines ConTracts ausschließlich auf explizite Benutzeranweisung – nicht automatisch
  - Regeln:
    - für jede(n) Step/Transaktion muß genau eine Kompensations-transaktion vorhanden sein
    - mit dem Commit des Steps müssen alle Daten, die für die Kompensation gebraucht werden, berechnet und abgelegt sein
    - lokale Daten, die für Kompensationsschritte benötigt werden, müssen bis End-Of_ConTract vor Löschen geschützt werden
    - nach der Entscheidung, einen ConTract zu kompensieren, müssen alle laufenden Steps abgebrochen werden bzw. es muß verhindert werden, daß weitere Steps gestartet werden
    - Kompensationen können auch mit Abort beendet werden, müssen aber dann wiederholt werden
    - keine (automatische) Behandlung des Falles, daß Kompensation nach k Wiederholungen immer noch nicht erfolgreich beendet werden konnte

- **Synchronisation mit *Invarianten***
  - Invarianten
    - Skript-Programmierer charakterisiert in Ausgangsinvariante den am Ende des Steps erreichten Zustand der bearbeiteten Objekte
    - Nachfolgender Step kann mit seiner Eingangsinvarianten überprüfen, ob die Bedingung für seine korrekte Synchronisation noch erfüllt ist
    - Beispiel einer Eingangsinvarianten eines Montage-Step: Anzahl der Schrauben im Magazin > 100
    - implizite Invarianzbedingung einer ACID-TA: Werte aller berührten Objekte bleiben bis TA-Ende eingefroren

➥ Serialisierbarkeit ist zwar hinreichend, aber oft keinesfalls notwendig für korrekten Ablauf

ConTracts (9)

- **Synchronisation mit *Invarianten* (Forts.)**
  - **Ziel**
    - Ermöglichen eines hohen Parallelitätsgrades und Ausschluß von Konsistenzverletzungen trotz frühzeitiger Sperrfreigabe (am Ende eines Si)
    - Invarianten steuern die Überlappung parallel ablaufender ConTracts bzw. Steps über Prädikate
  - Möglichkeiten



    - Sperrkonzept friert a zum Zeitpunkt e ein
    - Escrow erlaubt beliebige Änderungen im Unsicherheitsintervall
    - Check/reinvalidate-Prinzip erlaubt beliebige Änderungen von a; P(a) wird zum Zeitpunkt r neu evaluiert

Generalized Advanced Transaction Model

- **Concept of generalized advanced transaction**
  - T can be customized to different kinds of extended transaction models
  - by restricting the type of dependencies that can exist between the component subtransactions

- **Definition 1 (Subtransaction)**

  Subtransactions are low-level coordinated activities in an advanced transaction. A subtransaction T_i contains a set of related operations and forms a smallest coordination unit of the advanced transaction. The begin, abort and commit operations of subtransaction T_i are denoted by b_i, a_i and c_i respectively.

- **Definition 2 (Dependency)**

  In an advanced transaction model, a dependency specifies how the execution of primitives (begin, commit and abort) of a subtransaction T_i causes (or relates to) the execution of the primitives (begin, commit, and abort) of another subtransaction T_j.

- **Definition 3 (Generalized advanced transaction)**

  A Generalized Advanced Transaction T = <S,D> where S is the set of subtransactions in T and D is the set of dependencies between the transactions of T. We also use the term advanced transaction instead of generalized advanced transaction.

Generalized Advanced Transaction Model (2)

A set of dependencies has been defined in the work of ACTA[13]. Summarizing all these dependencies in previous work, we collect a total of sixteen different types of dependencies.

1. Commit dependency ($T_i \rightarrow_c T_j$): If both T_i and T_j commit, then the commitment of T_i precedes the commitment of T_j. Formally, $c_i \Rightarrow (c_j \Rightarrow (c_i < c_j))$.

2. Strong commit dependency ($T_i \rightarrow_{sc} t_j$): If T_i commits, then T_j also commits. Formally, $c_i \Rightarrow c_j$.

3. Abort dependency ($T_i \rightarrow_a T_j$): If T_i aborts, then T_j aborts. Formally, $a_i \Rightarrow a_j$.

4. Weak abort dependency ($T_i \rightarrow_{wa} T_j$): If T_i aborts and T_j has not been committed, then T_j aborts. Formally, $a_i \Rightarrow (\neg(c_j < a_i) \Rightarrow a_j)$

5. Termination dependency ($T_i \rightarrow_i T_j$): Subtransaction T_j cannot commit or abort, until T_i either commits or aborts. Formally, $e_j \Rightarrow e_i < e_j$, where $e_i \in \{c_i,a_i\}$, $e_j \in \{c_j,a_j\}$.

6. Exclusion dependency ($T_i \rightarrow_{es} T_j$): If T_i commits and T_j has begun execution, then T_j aborts. Formally, $(c_i \Rightarrow a_j) \vee (c_j \Rightarrow a_i)$.

7. Force-commit-on-abort dependency ($T_i \rightarrow_{fca} T_j$): If T_i aborts, T_j commits. Formally, $a_i \Rightarrow c_j$.

8. /9./10./11. Force-begin-on-commit/abort/begin/termination dependency ($T_i \rightarrow_{fbc/fba/fbb/fbt} T_j$): Subtransaction T_j must begin, if T_i commits (aborts/begins/terminates). Formally, $c_i\ (a_i/b_i/t_i) \Rightarrow b_j$.

12. Begin dependency ($T_i \rightarrow_b T_j$): Subtransaction T_j cannot begin execution, until T_i has begun. Formally, $b_j \Rightarrow (b_i < b_j)$.

13. Serial dependency ($T_i \rightarrow_s T_j$): Subtransaction T_j cannot begin execution, until T_i either commits or aborts. Formally, $b_j \Rightarrow (e_i < b_j)$ where $e_i \in \{c_i,a_i\}$.

---

13. Chrysanthis, P. , Ramamritham, K. : ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behavior, in: Proc. ACM SIGMOD Conf., 194-203, 1990

Generalized Advanced Transaction Model (3)

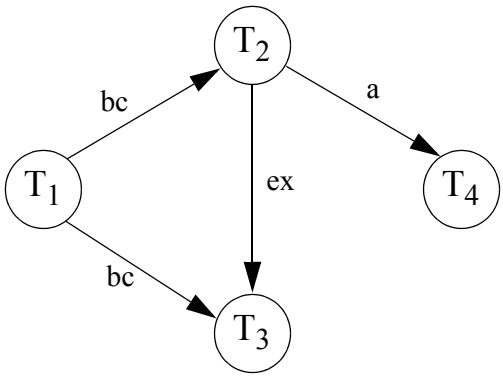
14. Begin-on-commit dependency ($T_i \rightarrow_{bc} T_j$): Subtransaction T_j cannot begin, until T_i commits. Formally, $b_j \Rightarrow (c_i < b_j)$.

15. Begin-on-abort dependency ($T_i \rightarrow_{ba} T_j$): Subtransaction T_j cannot begin, until T_i aborts. Formally, $b_j \Rightarrow (a_i < b_j)$.

16. Weak-Begin-on-commit dependency ($T_i \rightarrow_{wbc} T_j$): If subtransaction T_i commits, subtransaction T_j can begin executing after T_i commits. Formally, $b_j \Rightarrow (c_i \Rightarrow (c_i < b_j))$.

- **Example 1**

Consider an advanced application consisting of the following set of subtransactions $\{T_1,T_2,T_3,T_4\}$: [subtransactionT_1] - Reserve a ticket on Airline A; [subtransaction T_2] - Purchasing the Airline A ticket; [subtransaction T_3] - Canceling the reservation; and, [subtransaction T_4] - Reserving a room in Resort C. The various kinds of dependencies that exists among the subtransactions are shown the following Figure. There is a begin-on-commit dependency between T_1 and T_2 and also between T_1 and T_3. This means that neither T_2 or T_3 can start, before T_1 has committed. There is an exclusion dependency between T_2 and T_3. This means that either T_2 can commit or T_3 can commit, but not both. Finally, there is an abort dependency between T_4 and T_2. This means that if T_2 aborts, then T_4 must abort. We can model this application as an advanced transaction AT = <S,D>, where
$S = \{T_1,T_2,T_3,T_4\}$, $D = \{T_1 \rightarrow_{bc} T_2, T_1 \rightarrow_{bc} T_3, T_2 \rightarrow_{ex} T_3, T_2 \rightarrow_{a} T_4\}$.

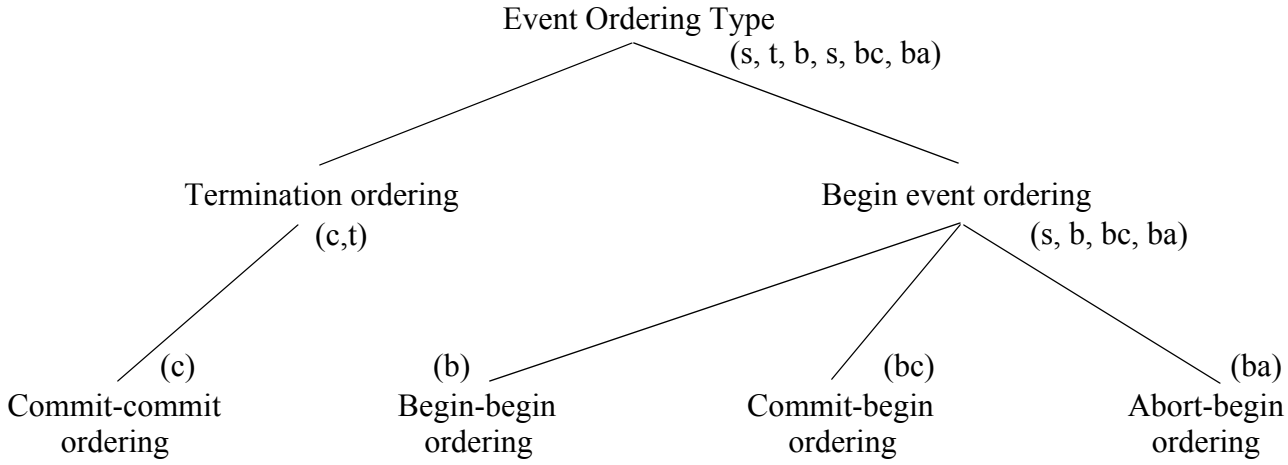


Generalized Advanced Transaction Model (4)

- **Definition 4 (Event Ordering)**

  An event ordering dependency $T_i \rightarrow T_j$ is one in which the execution of some event in subtransaction T_i must precede the execution of some event in subtransaction T_j.

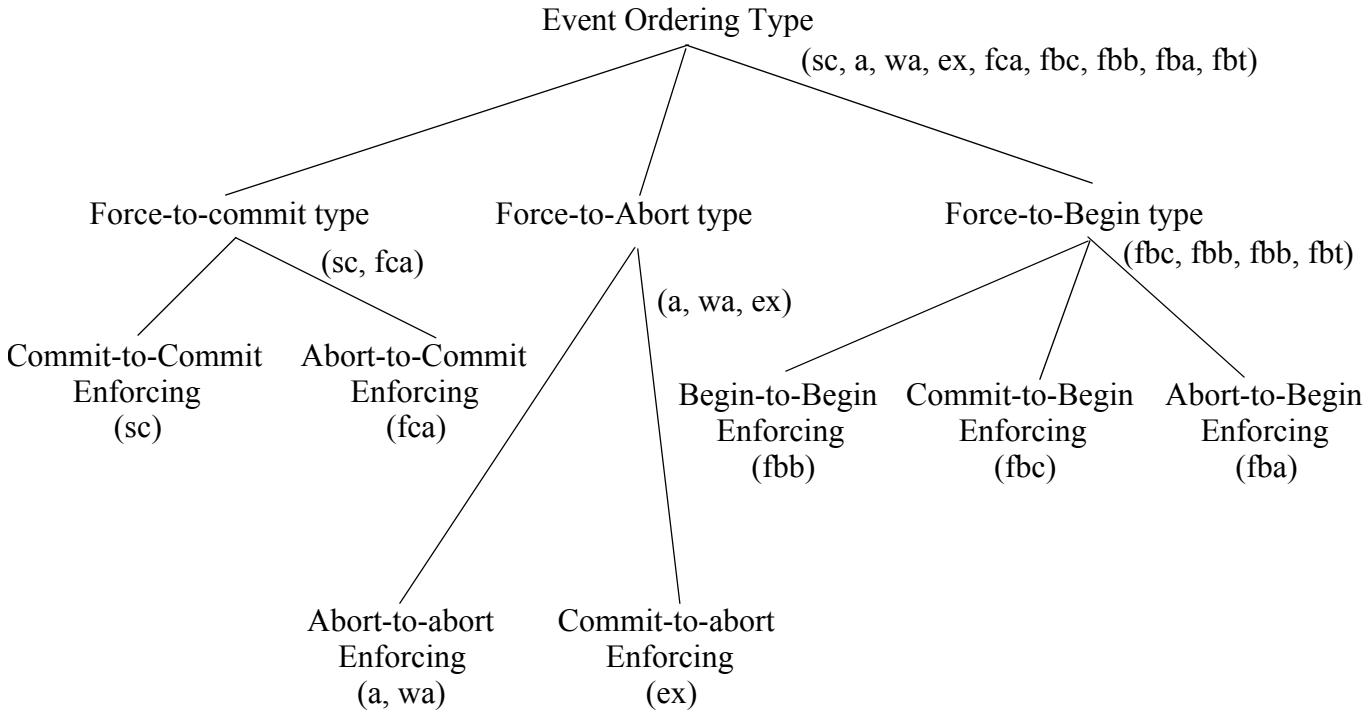
  The commit, termination, begin, serial, begin-on-commit and begin-on-abort dependencies are event-ordering dependencies. The event dependencies can be classified into ones that order the begin events and others that order the termination event. Detailed classification of the event ordering dependencies is given in the following Figure.



Generalized Advanced Transaction Model (5)

- **Definition 5 (Event Enforcement)**

  An event enforcement dependency $T_i \rightarrow T_j$ is one in which the execution of some event (begin, commit or abort) is subtransaction T_i requires the execution of some event in sub-transaction T_j.



Elektronischer Handel

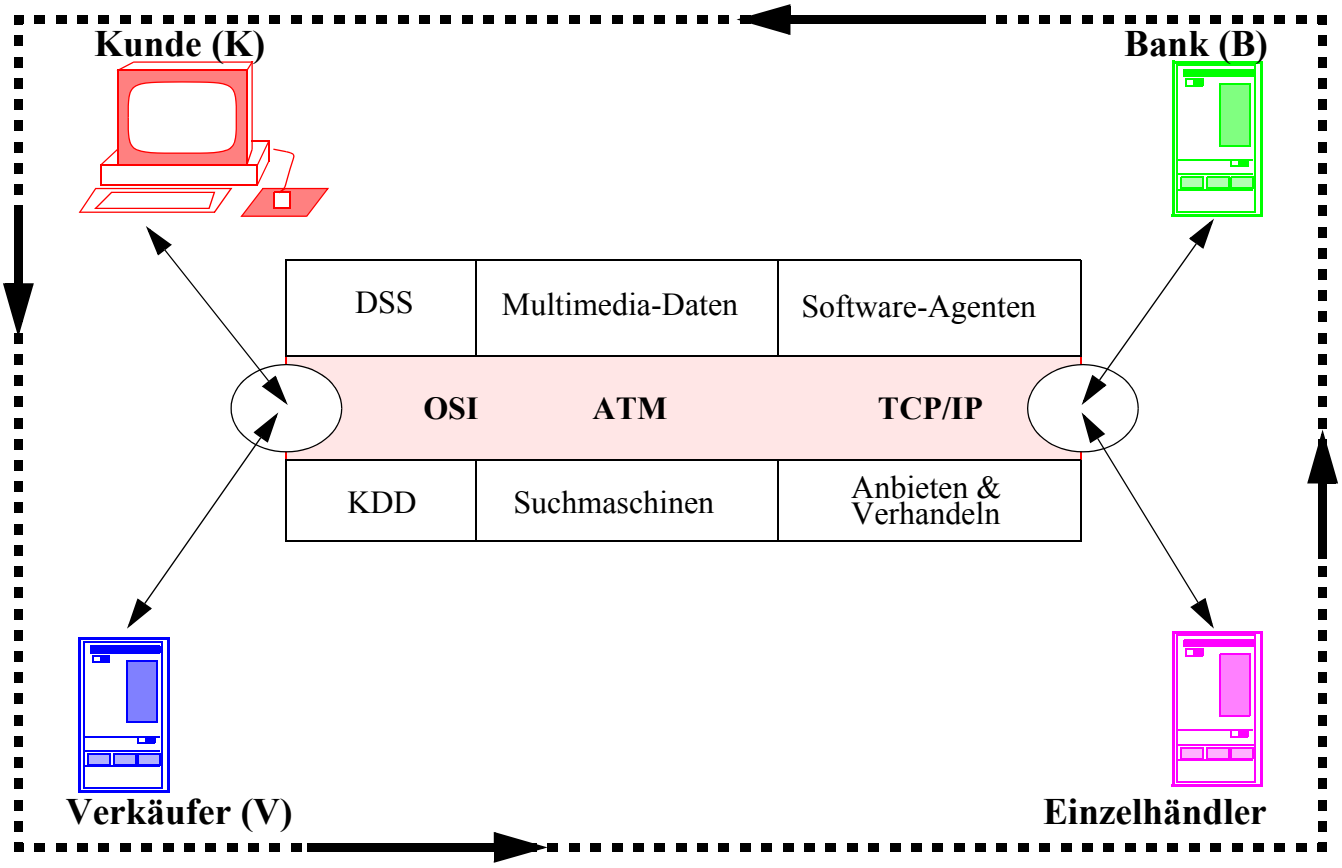
- **Situation**
  - Globalisierung, „Jeder kann jeden erreichen"
  - > $3*10^8$ Leute im Internet in 2003
  - Geschäftstransformation („Virtuelle Firma")
  - „Disaggregation" der Wertschöpfungskette (value chain)

- **Elektronischer Handel (EH)**

  „Wesentliche Elemente in der Interaktion zwischen Käufer und Verkäufer nutzen elektronische Medien so, daß entweder beim Käufer oder Verkäufer oder beiden keine natürlichen Personen involviert sind"

  (In der Regel ist mindestens der Verkäufer substituiert)

- **Konzeptuelles Modell des EH[14]**



14. DSS = *Decision Support System*,
KDD = *Knowledge Discovery in Databases*,
oft synonym zu Data Mining („Grabungstechniken für Wettbewerbsvorteile")

Elektronischer Handel (2)

- **Was ist anders beim EH?**
  - Geschäfts-/Einkaufsmöglichkeiten „rund-um-die-Uhr“
  - Unterstützung durch PCS-Systeme (price comparison shopper)
  - Veränderte Geschäftsmodelle – gezielte Kundenansprache

alle potentiellen Käufer

Unternehmen homogene Gruppe

Kunde A

  - **121**-Marketing
  - Beispiel: Buchhandel (4.8 Mio Bücher)

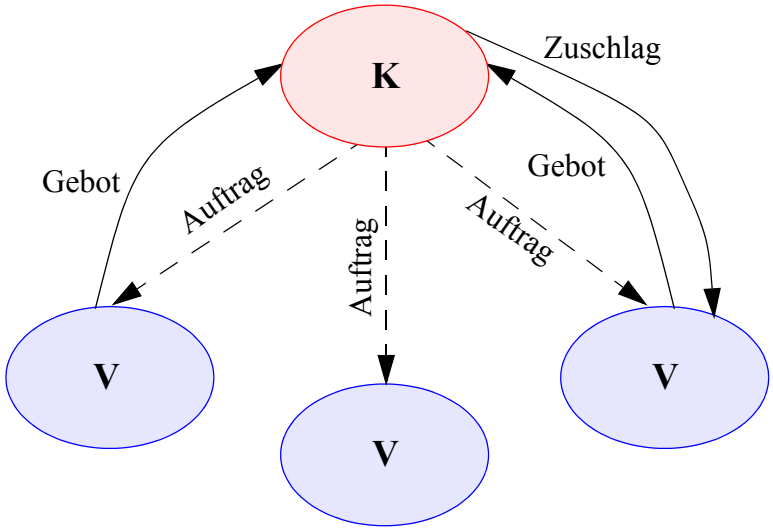
- **Anzahl der Verkäufer: Kunden**
  - 1 : 1 : Einzelgeschäft
  - n : 1 : Angebot
  - 1 : n : Auktion
  - n : m : Börse

- **Rechtliche Stellung**
  - Business-Business-Beziehung (B2B)
  - Business-Consumer-Beziehung (B2C)

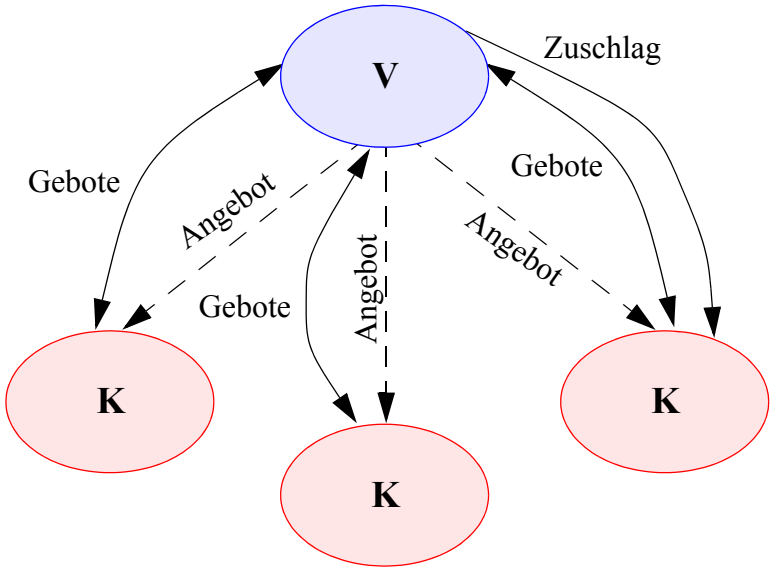
- **Abläufe beim EH**
  - im Internet, verteilte Verarbeitung (zwischen **K**, **V**, **B**)
  - Wie läßt sich Atomarität (und C’ I’ D’)
    bei verteilter Verarbeitung zwischen **K**, **V** und **B** erzielen?

Geschäftsmodelle beim EH

- **Angebotsprozeß**
  - Ablauf
    - Aufforderung zur Abgabe eines Angebotes
    - Abgabe der Gebote
    - Zuschlag
  - **Probleme**
    - Vertraulichkeit
    - Glaubwürdigkeit des Auftraggebers



- **Auktion**
  - Ablauf
    - Bekanntgabe des Angebotes
    - Abgabe der Gebote
    - Zuschlag
  - **Probleme**
    - Zeitrahmen für Zyklus von Angebot/Gebot/Zuschlag
    - Vertraulichkeit
    - Glaubwürdigkeit des Auktionators
    - Geschützter Raum der Auktion

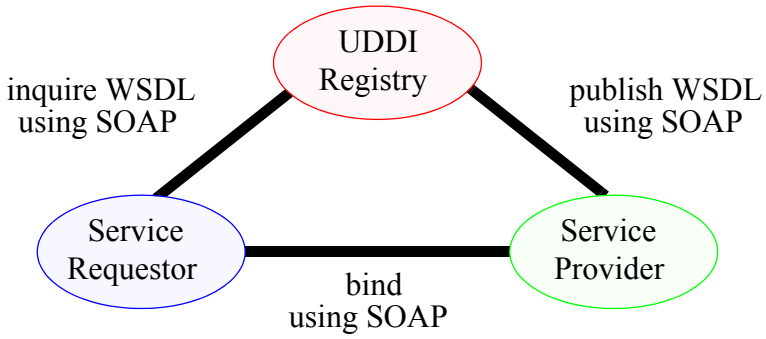


  - **Arten**
    - englische (up), holländische (down) Auktion
    - Vickrey-Auktion (second-price sealed-bit auction)
    - Höchstpreis-Auktion (first-price sealed-bit auction)
  - Formen: C2C-, B2C-, B2B-Auktion

Geschäftsprozesse und Web Services

- **Ziel: Prozeß- und Anwendungsintegration**
  - Daten- oder Informationsintegration
    - als komplexeste Form der Integration will die **semantische Vereinheitlichung** und Verknüpfung von Daten (Funktionsergebnissen) aus verschiedenartigen Quellen
    - wird hier nur innerhalb einzelner Aktivitäten eines Geschäftsprozessen verlangt
  - Prozeß- und AW-Integration
    - ist nicht so eng
    - will die Verknüpfung und Nutzung von **passenden Diensten in lose gekoppelten, verteilten und heterogenen** Umgebungen
    - baut sie als Komponenten, die koordinierte Aktivitäten ausführen müssen, in Geschäftsprozesse ein

- **Web Services[15] basieren auf**
  - **XML** als Kommunikationsformat
  - Web Service Description Language (**WSDL**) als XML-basierte Grammatik zur Beschreibung von Web Services
  - Simple Object Access Protocol (**SOAP**) als XML-basiertes Protokoll für den Austausch von Informationen in einer verteilten Umgebung
  - Universal Description, Discovery, and Integration (**UDDI**) als Standard für Geschäfts-Directories (Repositories mit Suchfunktionen) und ihre Web Services



15. A Web service (WS) is a software application identified by a URL, whose interfaces and binding are capable of being defined, described, and discovered by XML artifacts, and supports direct interactions with other software applications using XML-based messages via Internet-based protocols.

Geschäftsprozesse und Web Services (2)

- **Business Process Execution Language for Web Services (BPEL4WS)**
  - XML-basierte Definitionssprache für Abläufe
  - beschreibt Geschäftsprozesse, die WS sowohl bereitstellen als auch nutzen
    - Aktivitäten als „Steps“ im Kontrollfluß
    - Datenfluß
  - erlaubt die Spezifikation der WS-Koordination (WS können einen gemeinsamen Kontext mit Verknüpfungsinformation nutzen)
  - ermöglicht die Spezifikation von sog. Geschäftstransaktionen
    - Kontrollstruktur für die koordinierten WS-Aktivitäten
    - Eingriffsmöglichkeiten bei Fehlern
    - Rollback, Kompensationen

- **Geschäftsprozeß (business process)** definiert
  - potentielle Ausführungsreihenfolge der Aktivitäten (WS)
  - Daten, die von WS gemeinsam benutzt werden
  - Partner, die in den Geschäftsprozeß involviert sind
  - gemeinsame Ausnahme-/Fehlerbehandlung für Kollektionen von WS

- **Geschäftstransaktionen (business transactions, BTs)**
  - ACID-TAs (service atoms) als Bausteine, können geschlossen geschachtelte TAs sein (XA, 2PC)
  - Aggregationen von ACID-TAs, die Charakteristika und Verhalten von offen geschachtelten TA aufweisen (geschachtelte Sagas)
  - BTs sind nicht-atomar in dem Sinne, daß sie das selektive Commit und Abort von Teilnehmer-TAs erlauben
  - „Bemühenszusage“ bzgl. Rollback/Recovery

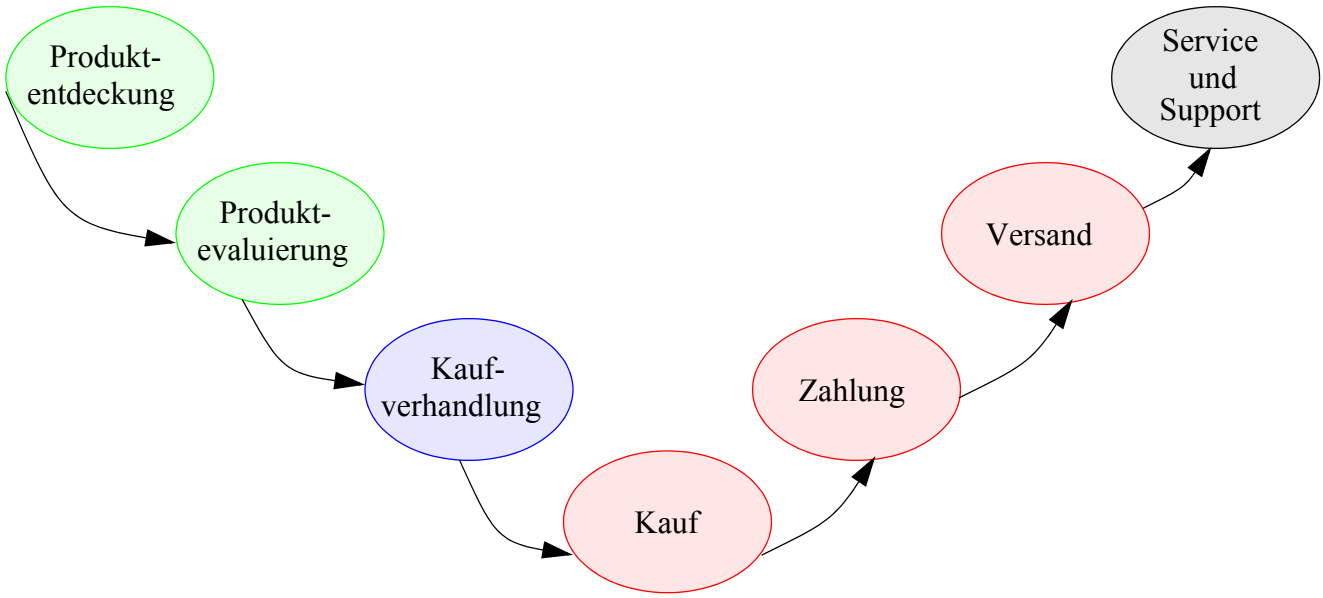
  ➥ Atomare WS für Kernaufgaben (z. B. Bestellung), nicht-atomare WS in Hilfsprozessen (Führung von Reisekosten-Konten)

Geschäftstransaktionen und Web Services

- **Bisher dominierende Eigenschaften**
  bei allen diskutierten TA-Modellen
  - Ausführung in geschützten Netzen (beyond firewalls)
  - TA-Monitor (WfMS) hat volle (ausreichende) Kontrolle über BM einer TA
  - Kooperationspartner (Teilnehmer) bekannt, meist eng gekoppelt

- **TAs in einer WS-Umgebung (BTs)[16]**
  - sind komplex
  - schließen mehrere (nicht-vertrauenswürdige) Teilnehmer ein
  - überspannen Organisationsgrenzen
  - sind langlebig

- **Geschäftsablauf beim EH (121)**

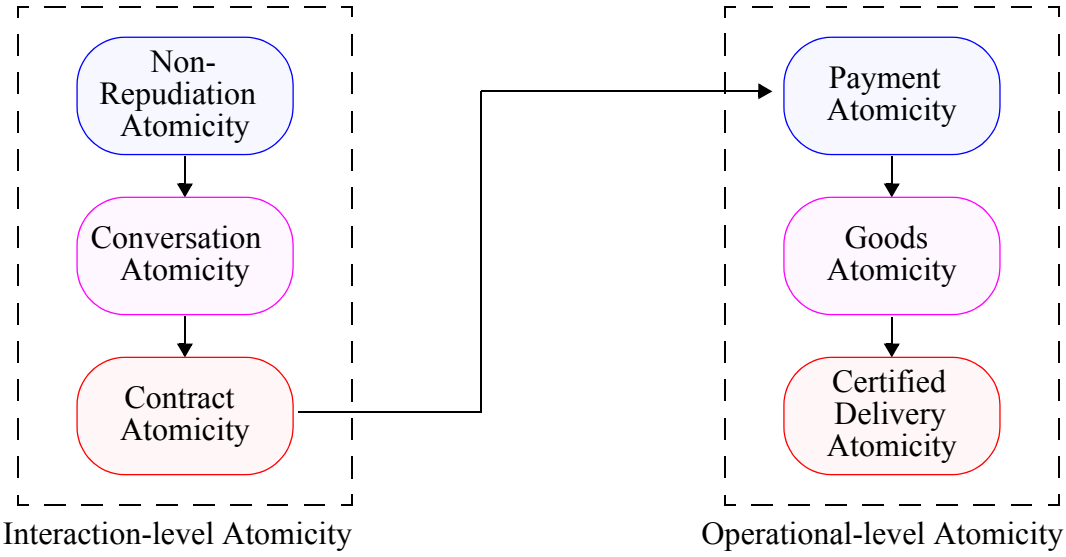


  - automatisierte Geschäftsabläufe mit Verhandlung, Vertragsabschluß, Lieferung mit Tracking, Bezahlung, Ausnahmebehandlung, ...
  - Welche TA-Eigenschaften können gefordert werden?

16. Papazoglou, M. P.: Web Services and Business Transactions, in: World Wide Web – Internet and Web Information Systems 6:1, Kluwer, March 2003, pp. 49-92

Geschäftstransaktionen und Web Services (2)

- **Erweiterung der klassischen TA-Verarbeitung**
  - „Unit of work“ einer BT soll die Semantik und das Verhalten einer Geschäftsaufgabe widerspiegeln
    - flexiblere Reaktionen auf Konfliktsituationen gefordert
    - Beispiel: BT regelt Kauf, Versicherung und Transport einer Ware
  - Kombination von kurzen transaktionalen[17] und langlebigen nicht-transaktionalen Prozessen in BTs
  - Verhalten wird von nicht-konventionellen Typen von Atomarität bestimmt



- **Atomaritäten auf Interaktionsebene (n Partner) hinsichtlich**
  - Verbindlichkeit (non-repudiation)
    Digitale Signatur aller Absprachen; Überprüfbarkeit durch Logging
  - Konversation
    Korrelation von Folgen von Anforderungen zwischen WS, allseitiges Rücksetzen innerhalb einer Konversation auf konsistenten Zustand
  - Vertrag
    Vertragsatomarität bindet gesetzlich n Partner; Verträge und Kontenbewegungen sind die Grundelemente von BTs

17. Atomarität einer Service-Anforderung bei einem Web Service: ACID-TA

Geschäftstransaktionen und Web Services (3)

- **Zahlungsatomarität (ZA)**
  - Transfer von „Geld“, wobei es keine Möglichkeit geben darf, Geld zu erzeugen oder zu vernichten
  - E-Card-Transaktionen besitzen typischerweise diese Eigenschaft

  ➥ **Grundlegende Eigenschaft,** die jedes BT-Protokoll erfüllen sollte

- **Warenatomarität (WA)**
  - Sie gewährleisten den vollständigen Austausch von „Geld und Waren“
  - Wenn **K** eine Ware mit Hilfe eines WA-Protokolls kauft, erhält er die Ware gdw das Geld überwiesen ist
  - WA ist besonders wichtig für „Informationsgüter“: Transfer kann unterbrochen werden; Unsicherheit auf beiden Seiten
  - **Analogie:** Lieferung eines Pakets per Nachnahme (cash on delivery)

  ➥ **Wichtige Eigenschaft,** die jedes BT-Protokoll zum Austausch von Informationsgüter erfüllen sollte

- **Zertifizierte Lieferung (ZL)**
  - **V** und **K** können genau überprüfen, welche Waren geliefert wurden
  - **Analogie:** Lieferung eines Paket per Nachnahme, wobei der Paketinhalt in Anwesenheit einer vertrauenswürdigen Person überprüft wird

  ➥ **ZL-Protokolle sind in Szenarien hilfreich,** in denen V und/oder K nicht vertrauenswürdig sind

- **Eigenschaften der BT-Atomaritätstypen**
  - Vertragsatomarität ➥ Konversationsatomarität ➥ Verbindlichkeitsatomarität
  - Zertifizierte Lieferung ➥ Warenatomarität ➥ Zahlungsatomarität

Geschäftstransaktionen und Web Services (4)

- **Weitere wünschenswerte Eigenschaft: Anonymität**
  - Gründe???
  - Käufe über $10.000 sind in der USA sofort zu melden
  - Oft wird eine begrenzte Form der Anonymität durch Einsatz eines Proxy-Agenten erzielt
  - **Analogie:** Kauf einer Ware von Automaten

- **Sicherheit**
  - Viele EH-Systeme hängen letztlich von einer vertrauenswürdigen Instanz (Autorität) ab
  - Unfälschbare und chiffrierte Aufzeichnungen (Logging) aller Geschäftsvorfälle in einem zentralen Server

- **Wert von Kreditkarten-Transaktionen**
  - Durchschnittlicher Wert: ~ $50
  - **V** zahlt an **B**: ~ 2% + 30 c
    ~ 2.25% + 50 c (bei „Mail Order“)

- **Was passiert bei „winzigen“ Transaktionen**
  - Mikrotransaktionen: < $1
  - Es gibt Bedarf, Transaktionen < 1 c abzurechnen (Millicent)

  ➥ **Schlüsselidee:** Aggregation von vielen kleinen TAs mit speziell optimierten Protokollen; Abbuchung innerhalb einer „großen“ TA

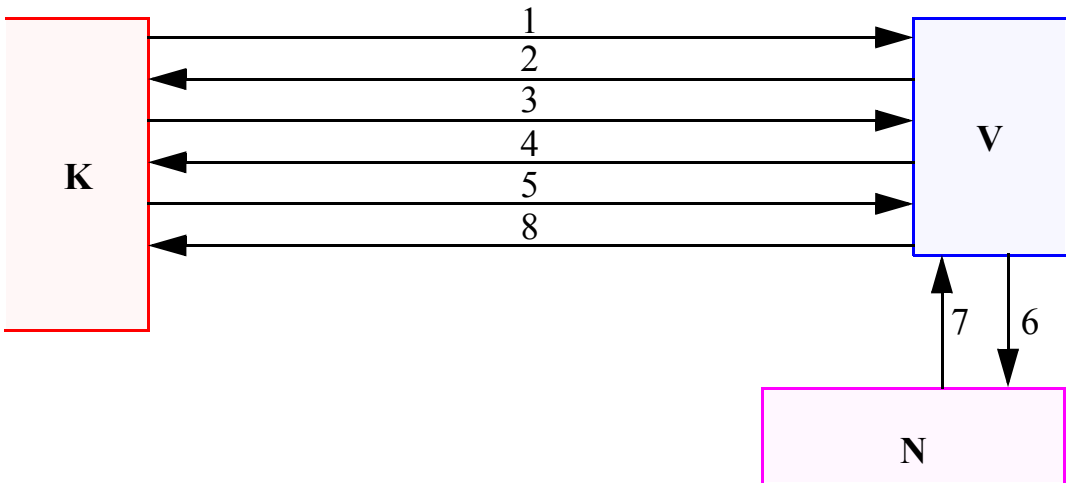
- **EH-Protokolle**
  - Digicash (nicht einmal ZA)
  - First Virtual
  - SSL
  - SET

  ➥ **Sie bieten nur die ZA-Eigenschaft**

NetBill-Protokoll

- **NetBill**
  - unterstützt alle drei Ebenen der Atomarität auf operationaler Ebene
  - läuft zwischen drei Parteien ab: **K**, **V** und **N** (NetBill-Server)
  - **N** fungiert als eine Art Notar (beiden Seiten bekannt)

- **Skizze des Protokolls**



1. **K** erfragt den Preis einer Ware bei **V**
2. **V** macht ein Angebot
3. **K** akzeptiert das Angebot
4. **V** schickt das mit **S** verschlüsselte Informationsgut W
5. **K** bereitet einen EPO (electronic purchase order) vor, der eine digitale Signatur für <Preis, chiffrierte Signatur des verschlüsselten W, Zeitstempel> enthält.
   K schickt den unterschriebenen EPO an **V**
6. **V** zeichnet den EPO gegen und signiert **S**. **V** schickt beide Werte an **N**
7. **N** überprüft die Signatur und Gegensignatur von EPO. **N** überprüft den Kontostand von **K** und den Zeitstempel von EPO. Wenn alles OK ist, transferiert **N** das Geld von **K**‘s auf **V**‘s Konto. **N** speichert den Schlüssel **S** und die chiffrierte Signatur von W. **N** bereitet eine signierte Empfangsbestätigung, die **S** enthält, vor und schickt sie an **V**
8. **V** speichert eine Kopie der Empfangsbestätigung und schickt sie an **K** weiter (K kann mit S die Ware W entschlüsseln)

NetBill-Protokoll (2)

- **Eigenschaften**[18]

  - **ZA**: Geldtransfer erfolgt nur im NetBill-Server

    ➥ **normale DB-Transaktion mit ACID-Eigenschaft**

  - **WA**: Commit-Punkt ist Schritt 7

    a) Fehler vor Schritt 7:
    Es erfolgt kein Geldtransfer und **K** erhält **S** nicht

    b) Fehler nach Schritt 7:
    N und **V** speichern **S**.
    K kann **S** entweder von **N** oder **V** erhalten, falls etwas schiefläuft

  - **ZL**:

    - Annahme:
      K behauptet, etwas anderes als das bestellte W erhalten zu haben
    - **N** besitzt chiffrierte Signatur des verschlüsselten W
      (von **K** und **V** gegengezeichnet)
    - **S** liegt bei **N** oder **V** vor

    ➥ **Schiedsstelle kann feststellen, daß K das chiffrierte W nicht verändert hat. Außerdem kann W nach Entschlüsselung mit Hilfe von S inspiziert werden.**

18. Tygar, J. D.: Atomicity versus Anonymity: Distributed Transactions for Electronic Commerce, in: Proc. 24th VLDB Conference, New York, 1998, pp. 1-12.

Zusammenfassung

- **Heterogene TA-Systeme**
  - Autonomie vs. Transparenz
  - schwierige und umständliche Lösung der Grundprobleme:
    globale Serialisierbarkeit, globale Atomarität,
    globale Deadlock-Erkennung
  - Einsatz von Agenten (Überwachung von „außen“)

- **Föderierte Mehrrechner-DBS**
  - lose (virtuelle) Integration unabhängiger, ggf. heterogener Datenbanken sowie von Anwendungssystemen
  - Nutzung einer generischen Anfragesprache (SQL)
  - Integration von Daten und Funktionen,
    auch über das Web möglich (Web Services)
  - Bildung von föderierten Funktionen,
    verschiedene Anbindungsmöglichkeiten

- **Eigenschaften von Workflows**
  - verteilt, langlebig, parallel, heterogen, hierarchisch organisiert
  - TA-geschützte und ungeschützte Aktivitäten
  - Workflow als globale TA? –
    nicht erreichbar, aber auch nicht wünschenswert

- **Anforderungen an die Wf-Ausführung**
  - Kosteneffektivität, Verlustminimierung im Fehlerfall
  - semantisch reichhaltigere Fehlerbehandlungsmodelle
    zwingend erforderlich
  - frühzeitige Freigabe von Betriebsmitteln (v.a. Daten),
  - Kompensation / Recovery oder manuelle Behebung im Fehlerfall

  ➥ **kein globales ACID, aber zumindest selektiv erforderlich**

Zusammenfassung (2)

- **Stratifizierte Transaktionen**
  - *Recoverable Messaging* als Grundlage asynchroner Transaktionsverarbeitung
  - Zerlegung der globalen TA in lokale TA, die in einzelnen baumverketteten Strata organisiert sind

- **ConTracts**
  - Beispiel für ‘transaktionale’ Workflows
  - ausschließliche Betrachtung von ACID-Transaktionen als Teil-Aktivitäten

- **Prozeß- und Anwendungsintegration**
  - Geschäftsmodelle und -Prozesse im Web
  - Beschreibung der Abläufe (Workflows): BPEL4WS
  - Schutz durch Geschäftstransaktionen (BTs), die offen geschachtelt sind und ACID-TA als „Bausteine“ benutzen (zur Sicherung unternehmens-kritischer Aktivitäten)

- **TAs in einer WS-Umgebung (BTs)**
  - sind komplex, mehrere (nicht-vertrauenswürdige) Teilnehmer
  - überspannen Organisationsgrenzen und sind langlebig
  - benutzen Web Services zur Realisierung (XML, WSDL, SOAP, UDDI)
  - neue Formen der Atomarität erforderlich: Verbindlichkeits-, Konversations-, Vertrags-, Zahlungs-, Warenatomarität sowie als komplexeste Form Atomarität bei zertifizierter Lieferung

  ➥ **Alle Konzepte beruhen auf Kompensationen, meist erforderlich wegen vorzeitiger Freigabe von Daten**