4. TA-Modelle für heterogene Systeme, Workflows und Web

- **Ziel**
  - Einführung in Daten-, Funktions-, Anwendungs- und Prozeßintegration
  - hier: Unterstützung durch TA-Konzepte
- **Heterogene TA-Systeme**
  - Autonomie vs. Transparenz
  - Probleme: Serialisierbarkeit, Atomarität, Deadlocks
- **TA-Verwaltung in heterogenen Systemen**
  - Korrektheitsbedingungen bei Mehrebenen-TA
  - Globale Serialisierbarkeit
- **Workflow-Management**
  - Ausführung von Workflows
  - Stratifizierte Transaktionen
  - ConTracts
- **Geschäftstransaktionen und Web Services**
  - Elektronischer Handel, Geschäftsmodelle
  - Einsatz von Web Services: BPEL4WS
  - Atomarität bei Geschäftstransaktionen

Autonomie in heterogenen Systemen

- **Entwurfsautonomie**
  - einzelne DBs sind unabhängig voneinander entworfen
  - variierender Grad an Autonomie
    - bei Eintritt in Verbund sind DB-Schema-Anpassungen möglich
    - lokale DB-Schemata bleiben unverändert erhalten
    - lokale DB-Schemata dürfen jederzeit „beliebig“ geändert werden

Â **Web Services[1] zielen auf diese extremste Form der Autonomie ab und verlangen deshalb eine sehr lose Form der Kopplung!**

- **Kommunikationsautonomie**
  - einzelne Komponentensysteme können selbst entscheiden, mit welchen anderen Systemen sie kommunizieren
  - autonomes DBS kann selbst bestimmen, wann es in den Verbund eintritt oder ihn verläßt
  - freie Entscheidung über die Art der Kommunikation (Verhandlung über Zugriff auf Datenbestände, Rolle von Import-/Export-Schemata)

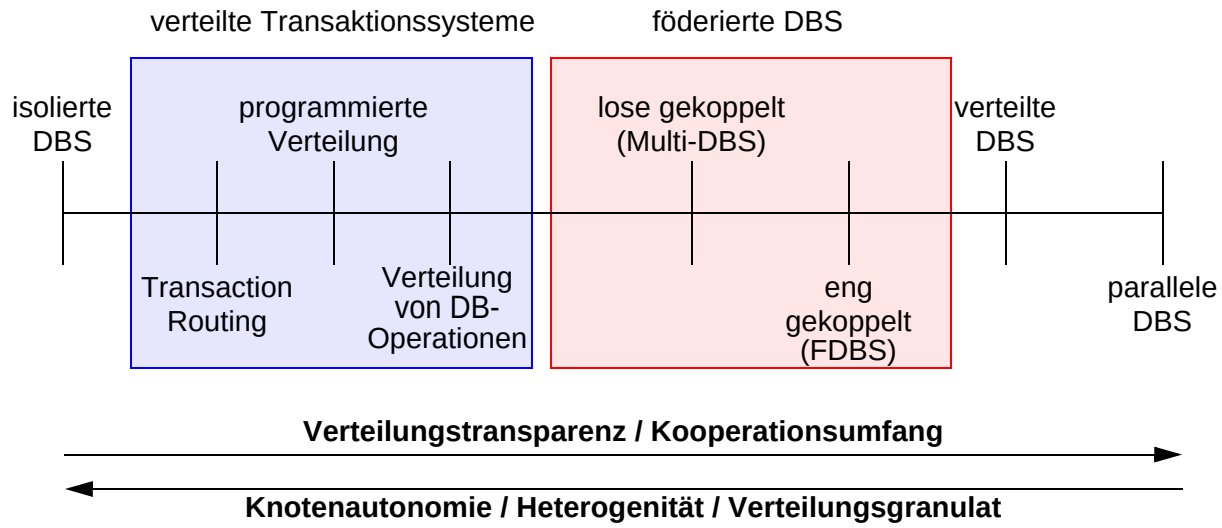
- **Ausführungsautonomie**
  - freie Entscheidung, welche Anwendungsprogramme, Anfragen und Änderungsoperationen ein System ausführt und
  - zu welchem Zeitpunkt es diese Operationen ausführt
  - gewisse Einschränkungen erforderlich, um sinnvolle Kooperation zu ermöglichen (Reihenfolge von Ops, 2PC, ...)

---

1. „The term Web Services refers to an architecture that allows applications to talk to each other“ (Adam Bosworth). Schlüsseleigenschaften ihrer Realisierung sind **Kommunikationseffizienz** (bei zustandslosen Protokollen), **lose Kopplung** (bei fein-granularem Zugriff auf gekapselte Objekte, die ihr Schema ändern dürfen), und **Asynchronität** aller Anforderungen.

Heterogene TA-Systeme

- **Autonomie vs. Transparenz[2]**

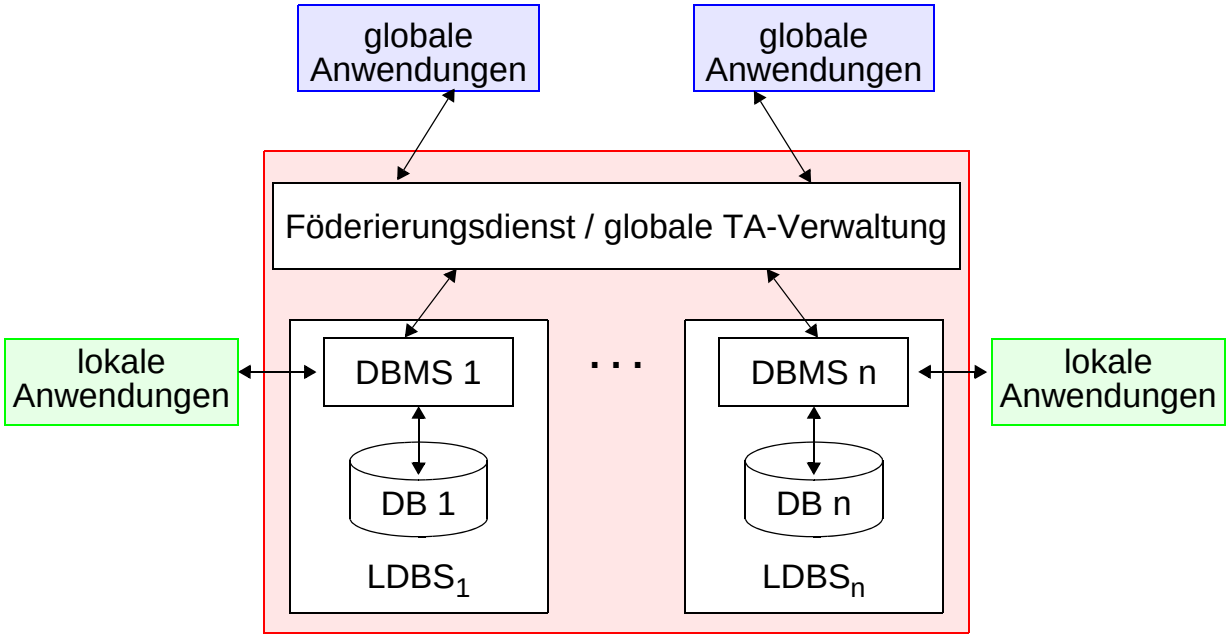


- **Verknüpfung von heterogenen Komponentensystemen durch**
  - **Föderierte DBS (FDBS)**
    - „engere“ Kopplung
    - oft homogenisierte Datensicht (externe Schemata) für globale Anwendungen
    - teilweise Aufgabe der Autonomie
    - Vor- und Nachteile von föderierten Systemen im Allgemeinen
  - **Multi-Datenbanksysteme (MDBS)**
    - „losere“ Kopplung
    - Wahrung der vollen Autonomie
    - keinerlei komponentenübergreifende Integritätsbedingungen
    - Zusammenschluß existierender DBS zu einem losen DBS-Verbund

2. Rahm, E.: Mehrrechner-Datenbanksysteme – Grundlagen der verteilten und parallelen Datenbankverarbeitung, Addison-Wesley, Bonn, 1994.

Heterogene TA-Systeme (2)

- **Grobarchitektur von FDBS und MDBS**



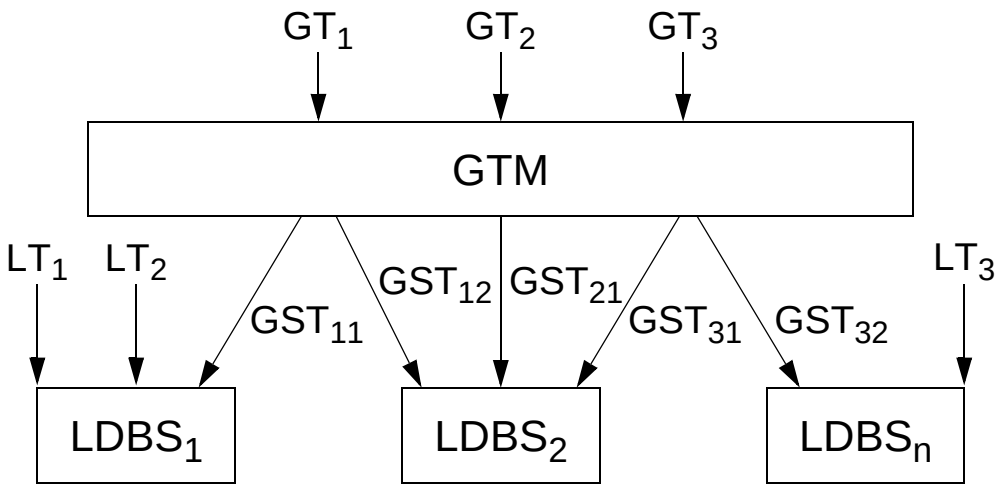
- Autonomie sehr wichtig, da Systeme zum Zeitpunkt der Integration bereits bestehen
- Föderierungsdienst / globale TA-Verwaltung erlauben eine Integration und Nutzung einer Menge von „isolierten“ und heterogenen DBS[3]
  - globale Anwendungen können auf alle Daten der beteiligten DBS zugreifen
  - lokale Anwendungen greifen, wie bisher auch, auf die Daten eines lokalen DBS zu

3. Eine mögliche Unterteilung liefern die folgenden Formen:
   - Heterogenität auf **Systemebene**, z. B. in Form unterschiedlicher Anfrage- und Zugriffsschnittstellen der beteiligten Systeme,
   - auf **Datenmodellebene** durch die Nutzung verschiedener Datenmodelle in den einzelnen Quellen
   - auf **Schemaebene** durch unterschiedliche Modellierungen eines Weltausschnitts,
   - auf **Datenebene** mit Konflikten aufgrund verschiedener Repräsentationen ein und desselben Real-Welt-Sachverhalts.

Heterogene TA-Systeme (3)

- **Struktur der Transaktionen**
  - systemübergreifende Transaktionen
    - Zerlegung in globale Sub-TA (GST) durch GTM
    - Ausführung der GST durch lokale DBS
  - lokale Transaktionen (LT)

- **Modell für TA-Abläufe**



  - GST werden durch LDBS wie LT behandelt

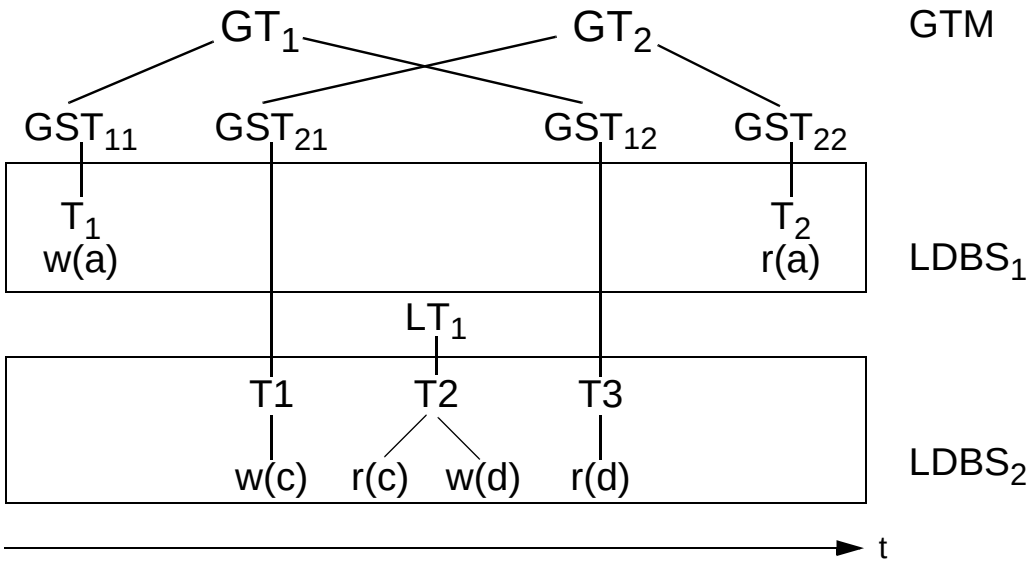
- **Jedes beteiligte DBS (LDBS) ist autonom**
  - kann im Prinzip eigenes Datenmodell besitzen
  - kann eigene Algorithmen für Synchronisation und Recovery einsetzen

Â **FDBS / MBDS sind, nicht nur auf Datenmodellebene, heterogene Systeme**

Heterogene TA-Systeme (4)

- **GTM kann nur GTs kontrollieren**
  - Annahme: alle Objekte, auf die durch GTs zugegriffen wird, können durch GTM identifiziert werden
  - Erkennung nur von direkten Konflikten zwischen GST_{ij} möglich

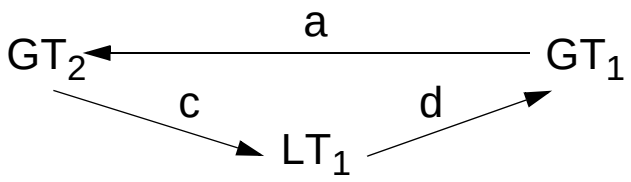
- **Beispiel**



  - erkannte Abhängigkeiten durch GTM

$$GT_1 \rightarrow GT_2$$

  - tatsächlicher Abhängigkeitsgraph



Heterogene TA-Systeme (5)

- **Lokale Autonomie impliziert**
  - Kontrollinformation eines LDBS (für Synchronisation und Recovery) wird nicht „nach außen“ verfügbar gemacht

  Â Globale Graphen für Serialisierbarkeitsprüfung oder Deadlock-Erkennung können nicht genutzt werden

  - Alle Zugriffe zu lokalen Daten (DB_i) werden durch das entsprechende $LDBS_i$ ausgeführt

  Â Globale TA erzeugen lokale Sub-TA für jedes $LDBS_i$, auf die sie zugreifen wollen

  - Ein LDBS kann nicht zwischen lokalen und globalen TA unterscheiden

  Â Sub-TA sind innerhalb eines LDBS nicht identifizierbar und werden nicht speziell behandelt

  - LDBS entscheiden unabhängig über das Commit lokal ausgeführter TA

  Â Selbst wenn das Abort einer Sub-TA einen nicht-serialisierbaren globalen Schedule ergibt, kann GTM das Commit dieser Sub-TA nicht erzwingen

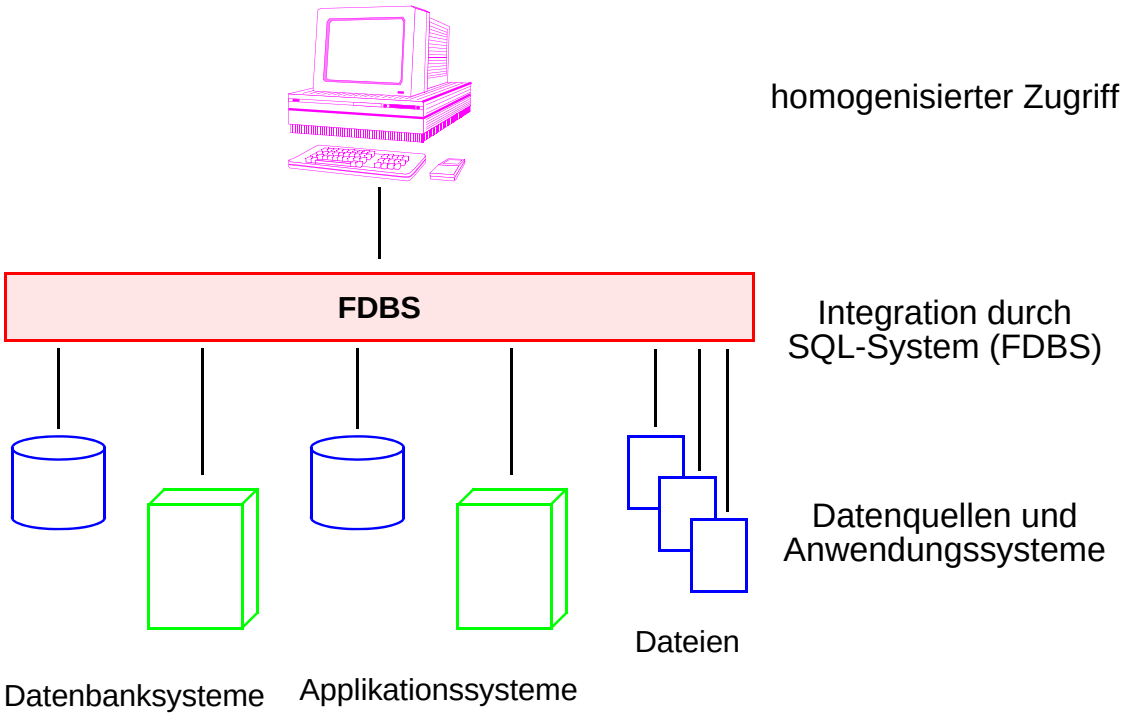
  - Bereits existierende lokale Programme werden nach der Integration weiterbenutzt

  Â Es ist keine Anpassung der SW (Schnittstellen) erforderlich

- **Heterogenität und Autonomie**
  erschweren die Erhaltung der globalen Konsistenz

Daten- und Funktionsintegration durch heterogene TA-Systeme

- **Ziel**
  - homogenisierter Zugriff auf eine Vielzahl verschiedenartiger Datenquellen
  - generische Anfrage- und Manipulationssprache (SQL)
  - aber: Anwendungssysteme kapseln ihre Daten und bieten nur einen funktionsbasierten Zugriff (API)



Â **Wie lassen sich TA-Eigenschaften bereitstellen?**

Daten- und Funktionsintegration durch heterogene TA-Systeme (2)

- **Klassifikation der lokalen Systeme**
  - Nicht wiederherstellbare Systeme (non-recoverable systems):
    Anwendungen auf Dateisystemen
  - Wiederherstellbare Systeme (recoverable systems)
    sind „nach innen“ transaktional
  - Transaktionale Systeme
    bieten einen sichtbaren Prepare-to-Commit-Zustand an
  - Offene transaktionale Systeme
    geben Informationen über TA-Zustände (Sperren, Deadlocks)
    nach außen

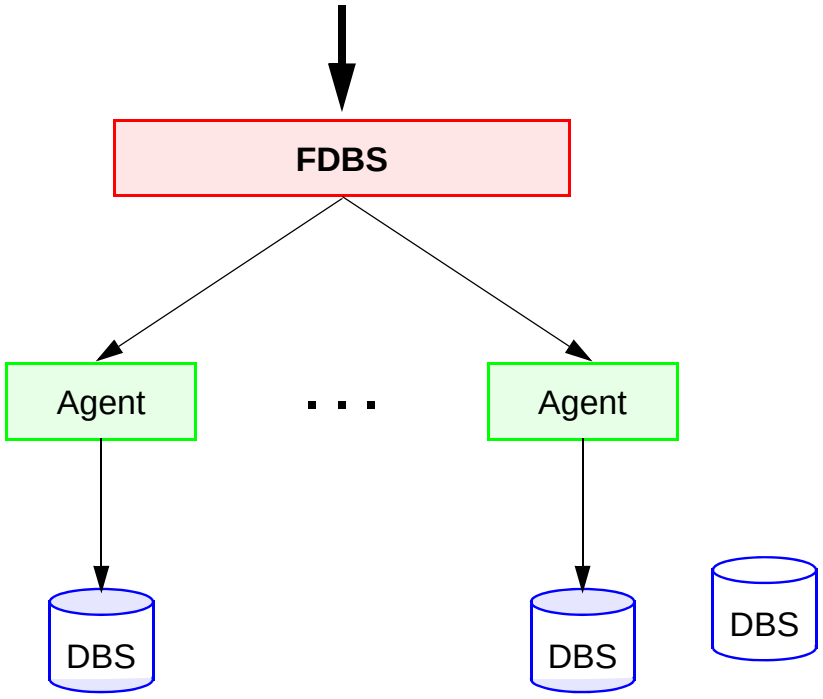
  Â **Einsatz von nicht-wiederherstellbaren Systemen verbietet sich für unternehmenskritische Daten und Anwendungen!**

- **Minimale Qualität der lokalen Systeme**
  - wiederherstellbar
  - Forderungen
    - an den Systemen keine Änderungen (Schnittstelle)
    - keine Rekompilation der Anwendungen
    - höchstens Erweiterungen (neue Tabellen) bestehender DB-Schemata

  Â **trotzdem:** umschließendes (globales) TA-Modell sollte Lösungen für **globale Serialisierbarkeit, globale Atomarität und globale Deadlock-Erkennung und -Vermeidung** besitzen.

Modell für mehrschichtige TA-Verwaltung

- **Dreischichtige FDBS-Architektur mit Agenten**



  - FDBS kontrolliert Ausführung globaler TA
  - lokale DBS führen lokale TA und Sub-TA globaler TA (globale Sub-TA oder kurz Sub-TA) aus
  - Agenten erlauben Kontrolle der Konflikte zwischen lokalen und globalen TA

- **TA-Verwaltung bei Mehrebenen-TA**
  - läßt sich flexibel an Gegebenheiten von FDBS anpassen
  - unterschiedliche TA-Verwaltungen in verschiedenen Schichten und sogar in der gleichen Schicht in verschiedenen Systemen möglich
  - Prinzip der strengen Schichtung kann teilweise aufgegeben werden: Sub-TA in den verschiedenen Systemen können sich über unterschiedlich viele Schichten aufteilen

  Â **Könnte anstelle eines lokalen DBS wieder ein FDBS auftreten?**

Korrektheitsbedingungen bei Mehrebenen-TA

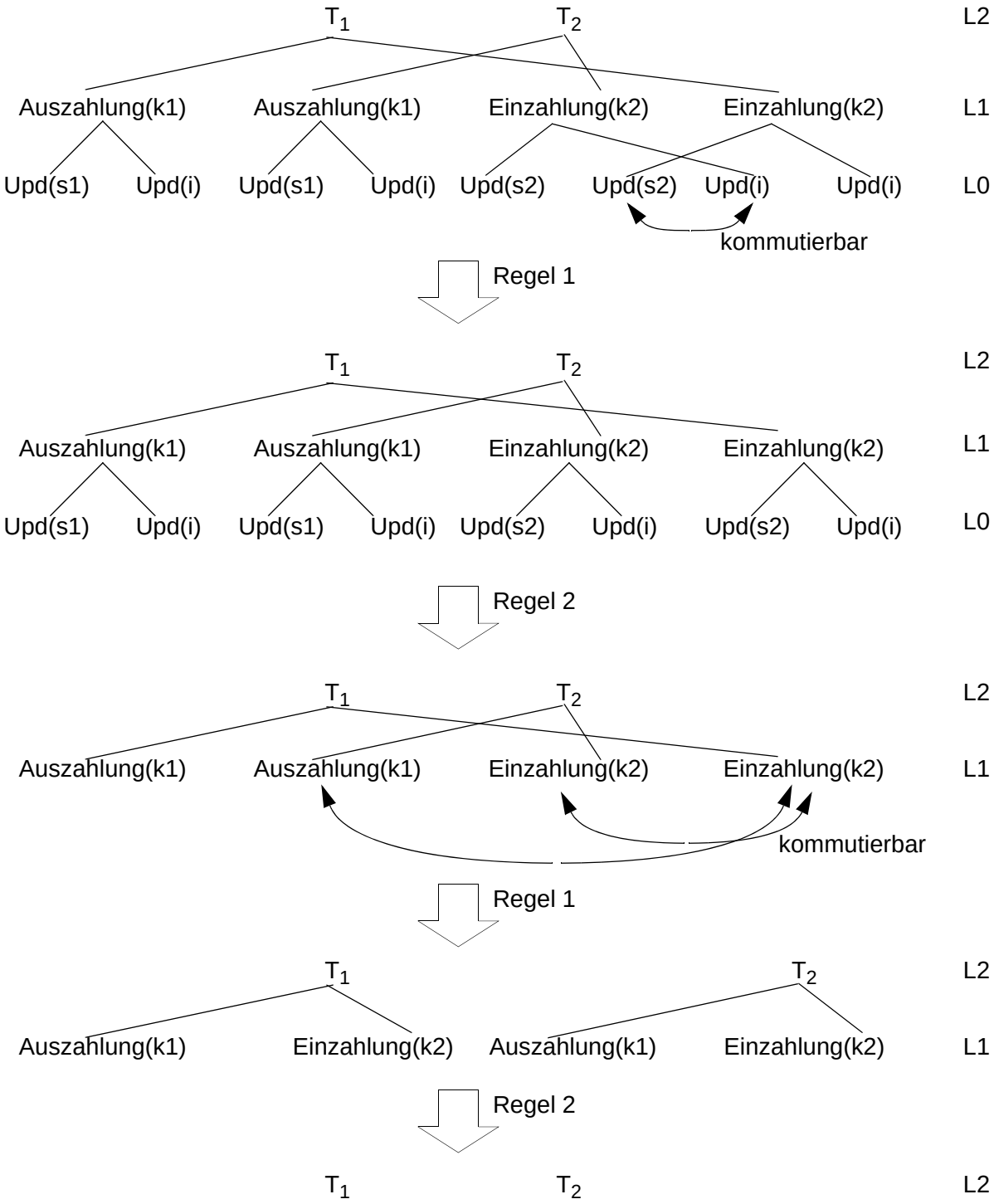
- **Globale Serialisierbarkeit**
  - kann durch Schicht-zu-Schicht-Serialisierbarkeit garantiert werden (**level-to-level serializability (L2L)**)
  - läßt sich überprüfen durch Umwandlung eines serialisierbaren Mehrebenen-Schedule in einen seriellen Schedule

- **Methode**
  - TA wird als Baum von geschachtelten Operationsaufrufen betrachtet
  - Isolierung von Teilbäumen
    Ein Teilbaum ist isoliert, wenn keine Überlappung mit anderen Teilbäumen auftritt, d.h., wenn im Schedule eine serielle Ausführung aller von der Wurzel des Teilbaums abhängigen Operationen stattfindet.
  - Reduktion von isolierten Teilbäumen

- **3 wichtige Regeln**
  - *Regel 1: Kommutativität von Operationen*
    Die Reihenfolge von zwei im zu überprüfenden Schedule benachbarten, geordneten Operationsaufrufen verschiedener TA kann vertauscht werden, wenn die Operationen kommutieren. Zwei Operationen o und o' kommutieren, wenn beide möglichen Ausführungsreihenfolgen o < o' und o'< o für sie selbst und für alle anderen nachfolgenden Operationen dasselbe Ergebnis liefern
  - *Regel 2: Reduktion von isolierten Teilbäumen*
    Ein isolierter Teilbaum von Operationsaufrufen kann bis auf die Wurzel reduziert werden
  - *Regel 3: Kompensation von Operationen*
    Wenn in einem Schedule eine Kompensationsoperation o^{-1} unmittelbar auf die Operation o selber folgt und beide Operationen isoliert sind, dann heben sich beide Operationen in ihrer nach außen sichtbaren Wirkung auf und sie können aus dem Schedule entfernt werden

Korrektheitsbedingungen bei Mehrebenen-TA (2)

- **Beispiel**

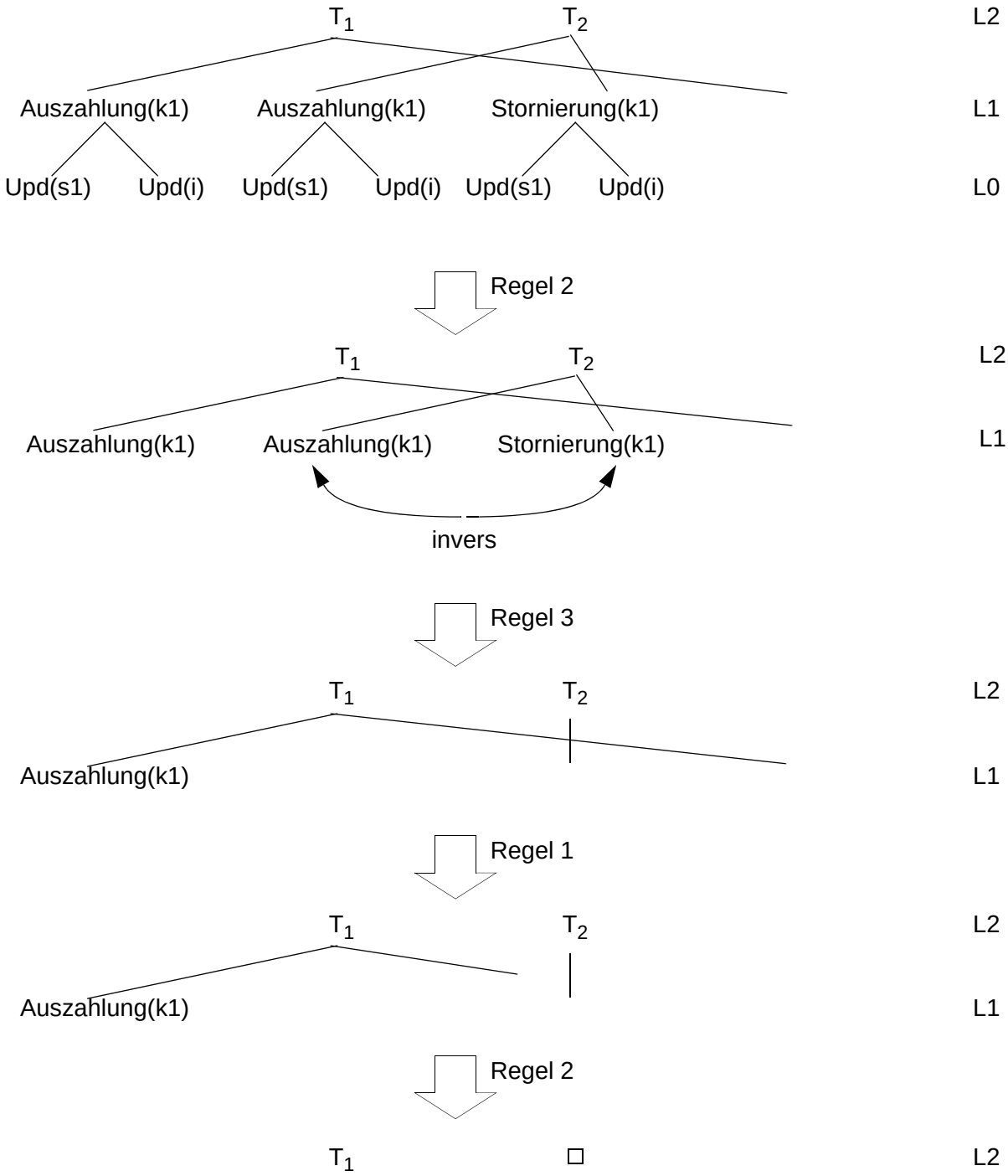


Â **Schrittweise Konvertierung in einen seriellen Schedule**

Korrektheitsbedingungen bei Mehrebenen-TA (3)

- **Beispiel**

  - *Regel 3: Kompensation von Operationen*

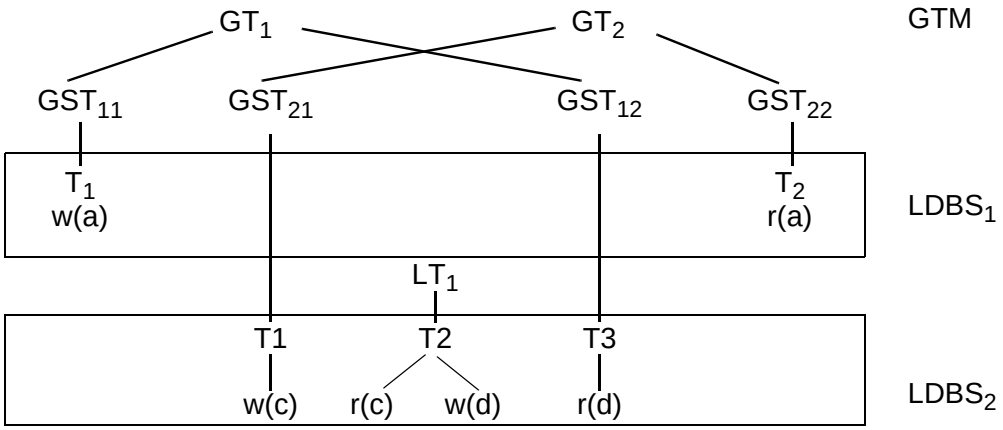


Â **Behandlung einer abgebrochenen Mehrebenen-TA**

TA-Verwaltung – Globale Serialisierbarkeit

- **Umsetzung der Korrektheitsbedingungen**
  bei heterogenen Systemen erforderlich

- **Globale Serialisierbarkeit**
  - überprüfbar für globale TA (ohne „unsichtbare“ Konfliktsituationen)



  - **aber:** was macht man bei lokalen TA und indirekten Konflikten?[4]

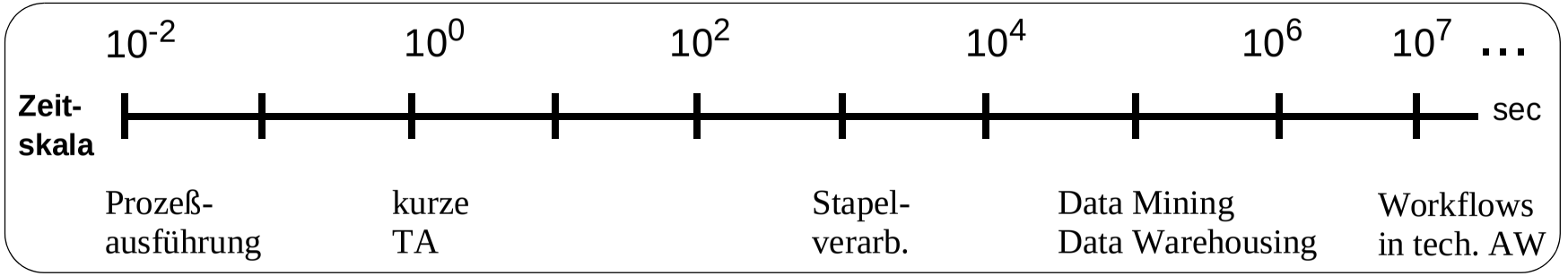
- **Erkennung und Vermeidung von indirekten Konflikten erfordert**
  - Konflikttests zwischen lokalen TA und globalen Sub-TA
  - spezielle Sperrmechanismen auf der Basis von zurückgehaltenen Sperren **(retained locks)**

- **Modifikation des Modells offener geschachtelter TA**
  - am Ende einer Sub-TA werden ihre Sperren nicht freigegeben, sondern zurückgehalten
  - normale Sperren gelten für alle TA
  - zurückgehaltene Sperren werden nur bei lokalen TA angewendet

4. Es gelten weiterhin die Annahmen: Lokale Systeme liefern keine Informationen über ihre TA-Verwaltung, und sie können auch nicht modifiziert werden

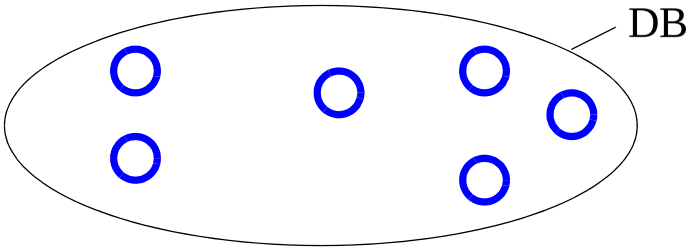
Workflows

- **Neue Herausforderung – Dauer der Ablaufkontrolle**

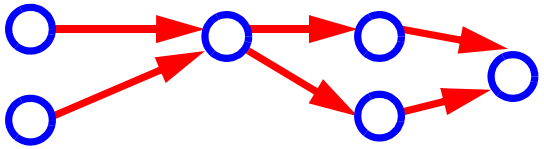


- **Strukturierung der Abläufe: Abhängigkeiten auf gemeinsamen Daten**

flache Transaktionen besitzen keine Struktur



Workflows müssen Struktur aufweisen

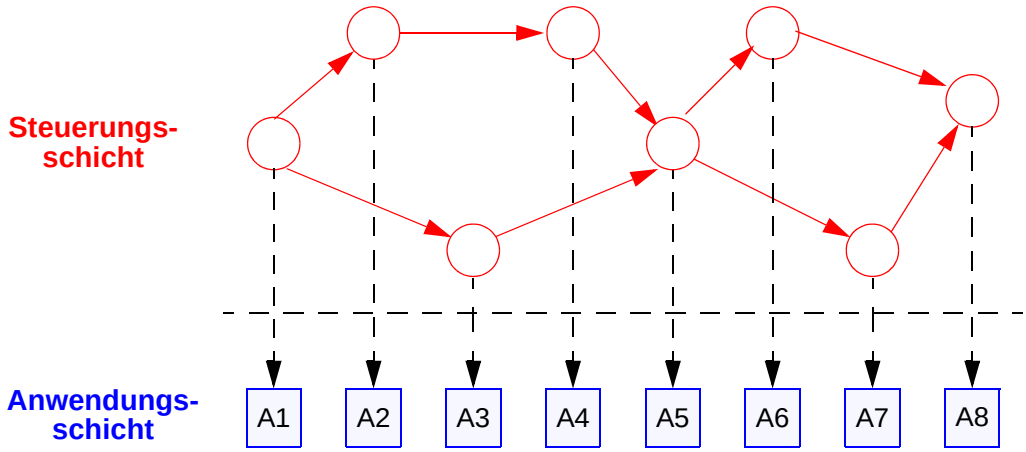


Â **transaktionsgestützte Workflows**

Ausführung von Workflows

- **Definitionsversuche:**
  - Ein ***Geschäftsprozess*** ist eine Menge von manuellen, (teil)automatisierten betrieblichen Aktivitäten, die nach bestimmten Regeln auf ein bestimmtes (unternehmerisches) Ziel hin ausgeführt werden
  - ***Workflow*** is a collection of activities performed by information systems and/or humans to carry out a business process.
  - ***Workflow-Management*** supports integrated definition, validation, analysis, enactment, and monitoring of processes in a heterogeneous environment.

- **Eigenschaften von Workflows**
  - verteilt, parallel
  - langlebig[5]
  - heterogen, hierarchisch organisiert
  - Aktionen auf getrennten und gemeinsamen Datenbereichen (Wf- und AW-spezifisch)
  - TA-geschützte und ungeschützte Aktivitäten (Ai)
  - Kooperation zwischen unabhängigen Komponenten (z. B. RM)

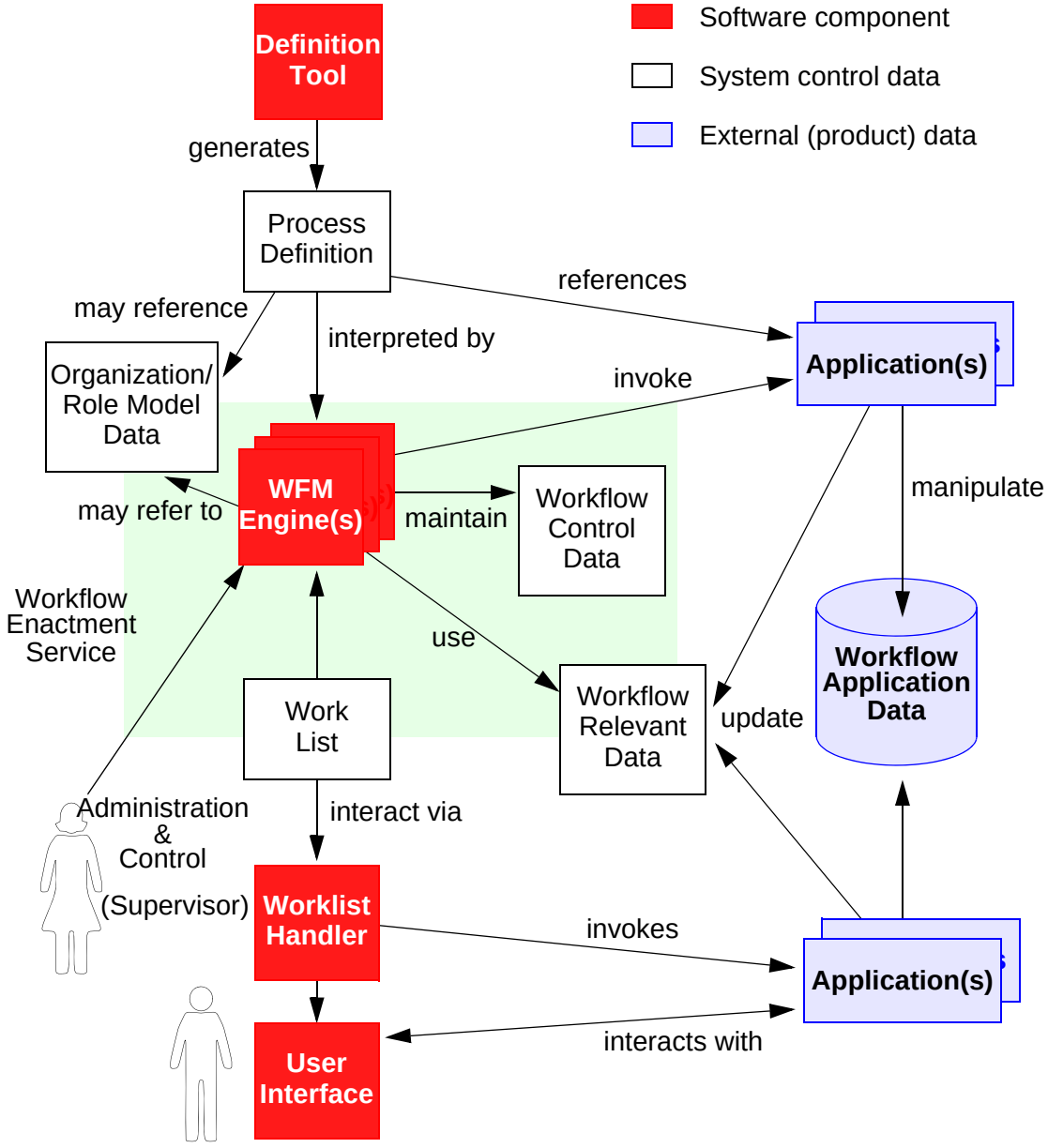


5. Bei Workflows und Geschäftsprozessen (insbes. im Web) ist das meist synomym zu „people-driven“. Wfs für Entwurfsvorgänge heißen auch „Designflows“.

Ausführung von Workflows (2)

- **Überblick über Aufgaben und Abhängigkeiten**

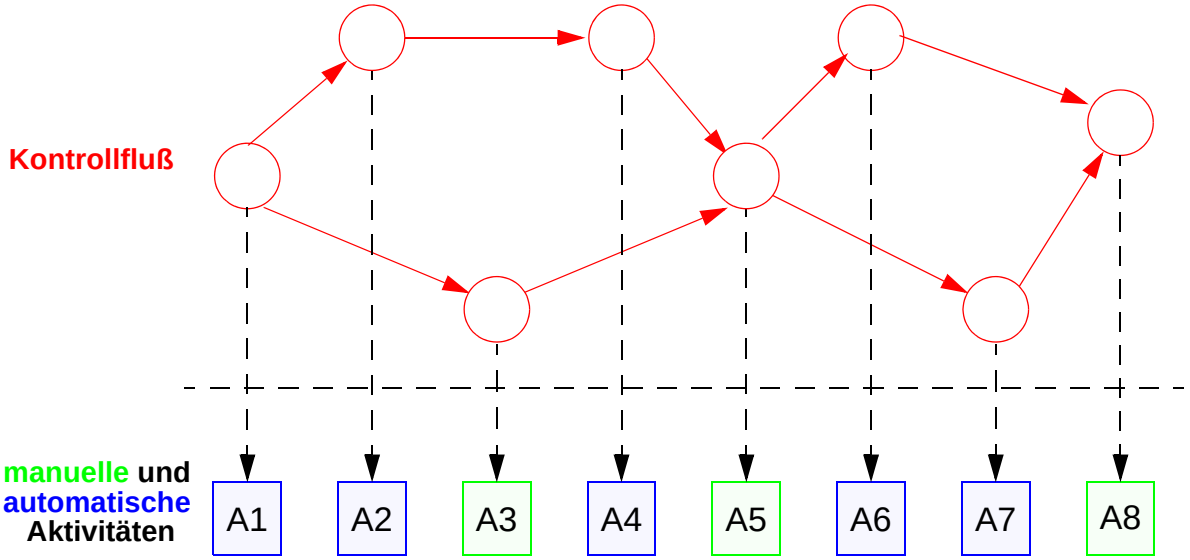
  eines Workflow-Management-System nach WfMC[6]



6. Workflow Management Coalition: http://www.wfmc.org/standards/docs.htm

Ausführung von Workflows (3)

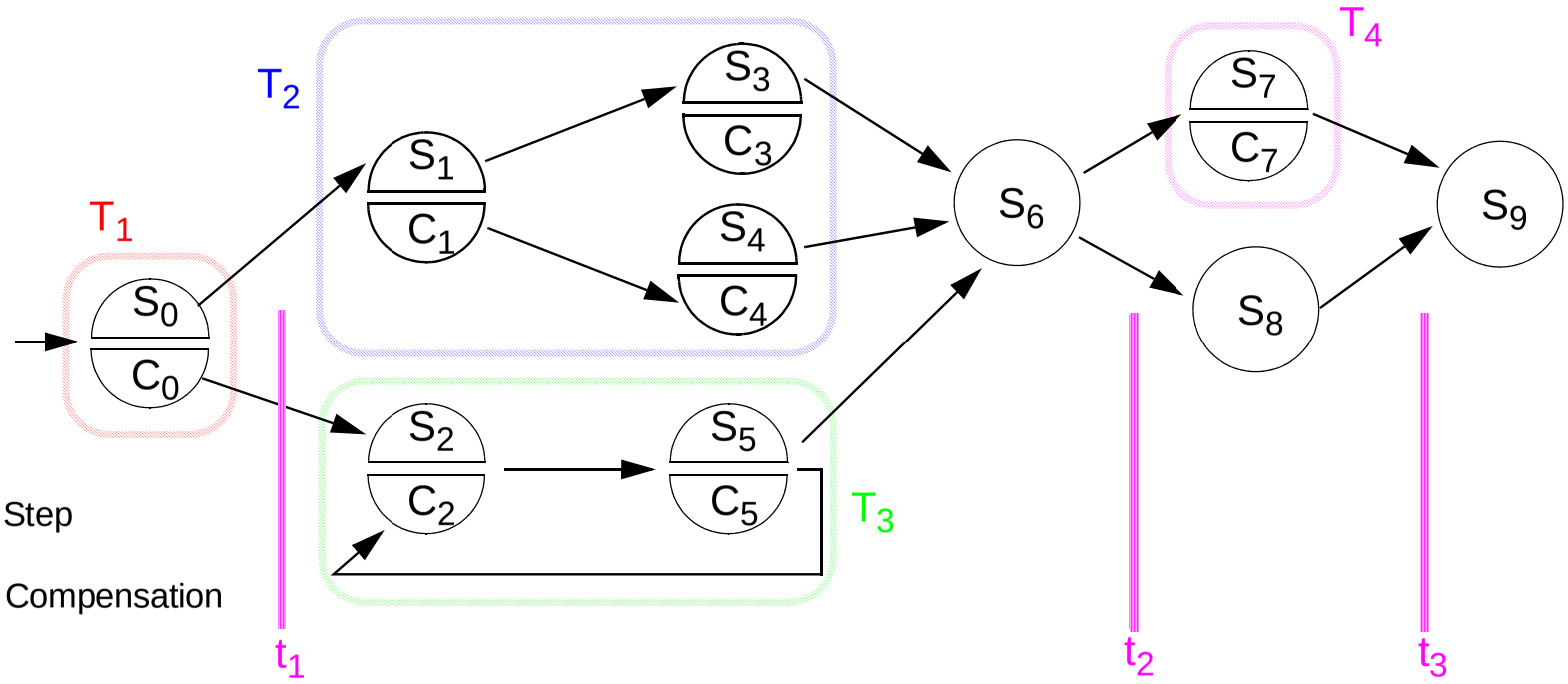
- **Zwei Arten von Datenhaltung erforderlich**
  - Kontrolldaten (durch WfMS (-DBMS) verwaltet)
  - Produktionsdaten (durch AW oder AW-DBMS verwaltet)
  - keine abgestimmten Kooperationskonzepte (Autonomie, Heterogenität)
  - **Achtung:** WfMS und AWs sind aus der Sicht ihrer DBMS Applikationen!



- **Konsequenzen**
  - ACID und damit einfache Fehlersemantik sind nicht zu gewährleisten
  - Atomarität von Workflows nicht erreichbar, aber auch nicht wünschenswert
  - bestimmte Aktivitäten (Steps) können nicht wiederholt werden
  - Transaktionsschutz (ACID) zumindest selektiv erforderlich
  - kritische Kooperationen erfordern TA-Schutz durch RM-Protokolle

Ausführung von Workflows (4)

- **Transaktionsgestütztes Workflow-Szenario – allgemeine Problemstellung**



Â **Recovery-Aktionen bei**

- Rücksetzen von T_2
- Crash bei t_1, t_2, t_3

- **Was passiert mit den ungeschützten Aktionen?**

Ausführung von Workflows (5)

- **Crash in Workflows**
  - offene Transaktionen werden bei Crash zurückgesetzt
  - abgeschlossene Transaktionen bleiben vom Crash unbeeinflußt; bei Rollback des Workflow müssen sie, wenn sie betroffen sind, vollständig kompensiert werden
  - Aktionen, die nicht dem Transaktionsschutz unterliegen, gehen verloren

- **Voraussetzungen für Crash-Recovery**
  - alle persistenten Statusinformationen aller aktiven Workflows können für Vorwärts-Recovery (Fortführung des Wf) benutzt werden
  - persistente Kontexte können einer Anwendungsfunktion wiederholt zur Verfügung gestellt werden
  - persistente Ausführungshistorie gestattet ein Rollback mit zeitlich gestaffelten Aufsetzmöglichkeiten

- **Semantisch reichere Fehlermodelle erforderlich**
  - Es geht immer nach vorne!
    Ein **„Zurück“ ist eigentlich** nicht vorgesehen
  - frühzeitige Freigabe von Änderungen auf gemeinsamen Daten
  - Verwaltung der Ausführungsgeschichte und der Kontextdaten erlauben Unterbrechbarkeit sowie eine gewisse Art von Vorwärts-Recovery
  - **persistente** Zwischenergebnisse und Nachrichten

Â Kombination von TA, persistenten Warteschlangen und Kompensation bei Blokkierung, Wiederanlauf sowie Rücksetzen von TA-Ergebnissen

Stratifizierte Transaktionen

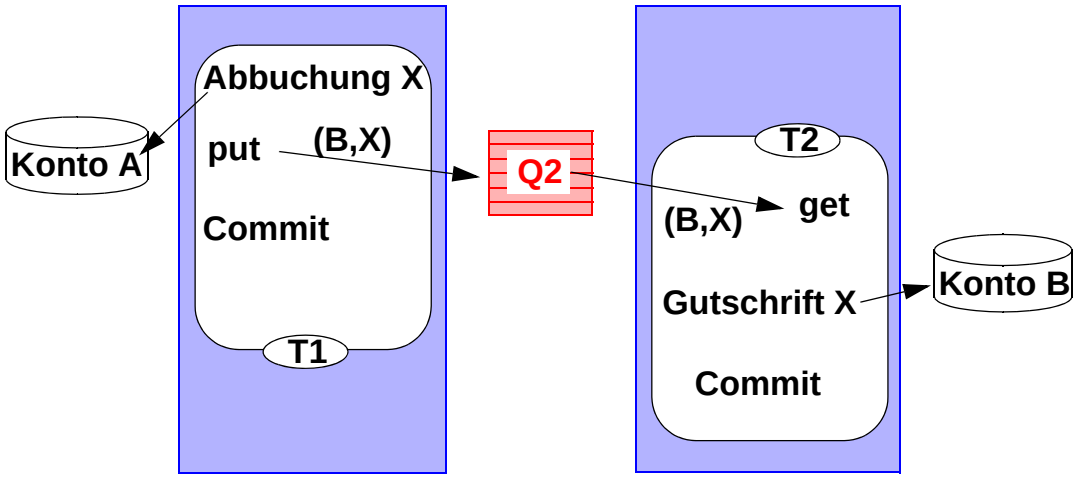
- **Phoenix-Verhalten von Workflows**
  - Recoverable Messaging erlaubt die Nutzung persistenter Warteschlangen (Queued Transaction Processing)

  Â Absicherung der Kommunikation zwischen WfMS-Komponenten

  - automatisierter Wiederanlauf ohne „Verlust eines Arbeitsschritts“
  - Gruppen von TAs mit 2PC-Abschluß: Konzept der stratifizierten TAs

  Â Anbindung von Anwendungs-TAs an systeminterne TAs des WfMS (Sicherung der Kontexte im WfMS-DBS)

- **Einschub „Recoverable Messaging“**



Zerlegung der globalen Transaktion “Buchung von Konto zu Konto”

- Wichtigstes Prinzip: Enqueue / Dequeue wird jeweils innerhalb der Kontrollsphäre der Schreib- / Lesetransaktion durchgeführt
- erfordert entsprechende Protokolle zwischen Queue-Manager und TA-Manager (mindestens 2PC)
- Grundlage asynchroner Transaktionsverarbeitung
- MOM: Message-oriented Middleware

Stratifizierte Transaktionen (2)

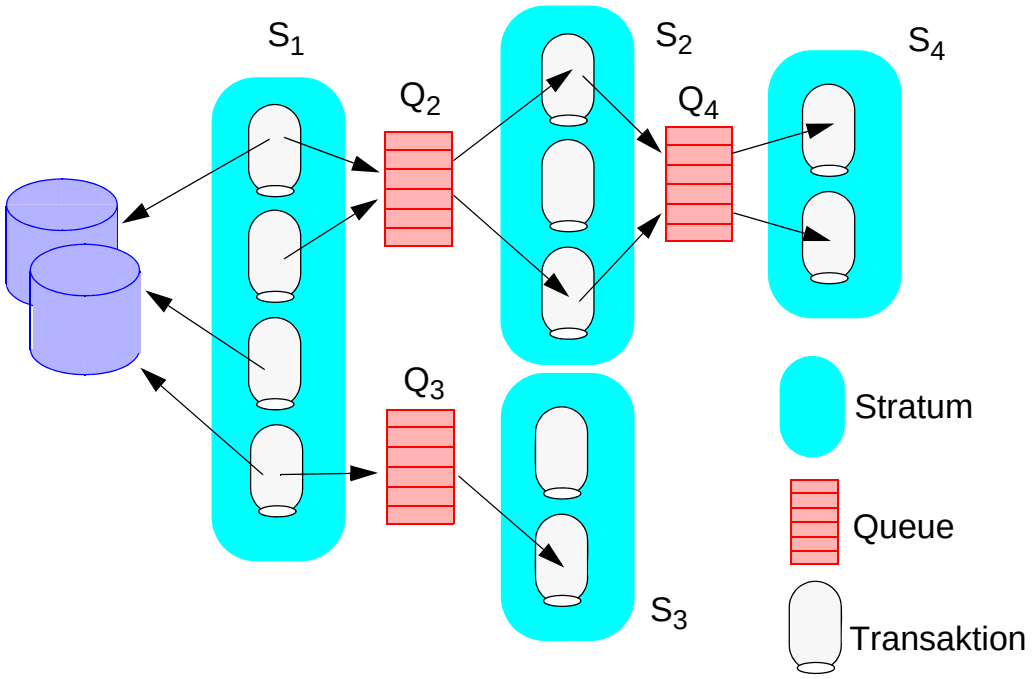
- **AW-orientierte Zerlegung der Transaktion T**
  - Erweiterung des Konzepts der Verkettung von TA durch Recoverable Messaging
  - Zerlegung in (möglicherweise verteilte) T_1, ... T_n;
  - Verkettung: jede T_i erhält persistente Warteschlange Q_i, aus der sie Anforderungen erhält, bestimmte Operationen auszuführen
  - lineare Reihenfolge nicht zwingend

- **WICHTIG:**
  - Alle von den Transaktionen T_i manipulierten Ressourcen (also insbes. auch die Nachrichten) sind wiederherstellbar
  - Dies bedeutet, daß sich die von den Transaktionen T_i benutzten RM (DBMS, MOM) in atomares Commit einbinden lassen (XA, 2PC)

- **Struktur stratifizierter Transaktionen**
  - Einige T_i sollen gemeinsam zum erfolgreichen Ende kommen
  - disjunkte Zerlegung von T in Transaktionsmengen S_1, ..., S_m
  - ($S_i \subseteq T$ mit $S_i \neq 0$, und $S_i \cap S_j = 0$ für $i \neq j$, und $U_{j=1}^{m} S_j = T$)
  - Transaktionen von S_i werden durch 2PC-Protokoll synchronisiert
  - Menge S_i von Transaktionen heißt ***Stratum***

Stratifizierte Transaktionen (3)

- **Verkettung der Strata**
  innerhalb der stratifizierten Transaktion T durch Baumstruktur



- **Alle Strata führen schließlich Commit aus**
  unter der Bedingung, daß die jeweiligen Vater-Strata
  zu irgendeinem Zeitpunkt vorher Commit ausgeführt haben

- **Falls Stratum wiederholt scheitert (echte Ausnahme):**
  stratifizierte Transaktion muß zurückgesetzt werden (Kompensation)

ConTracts (1)

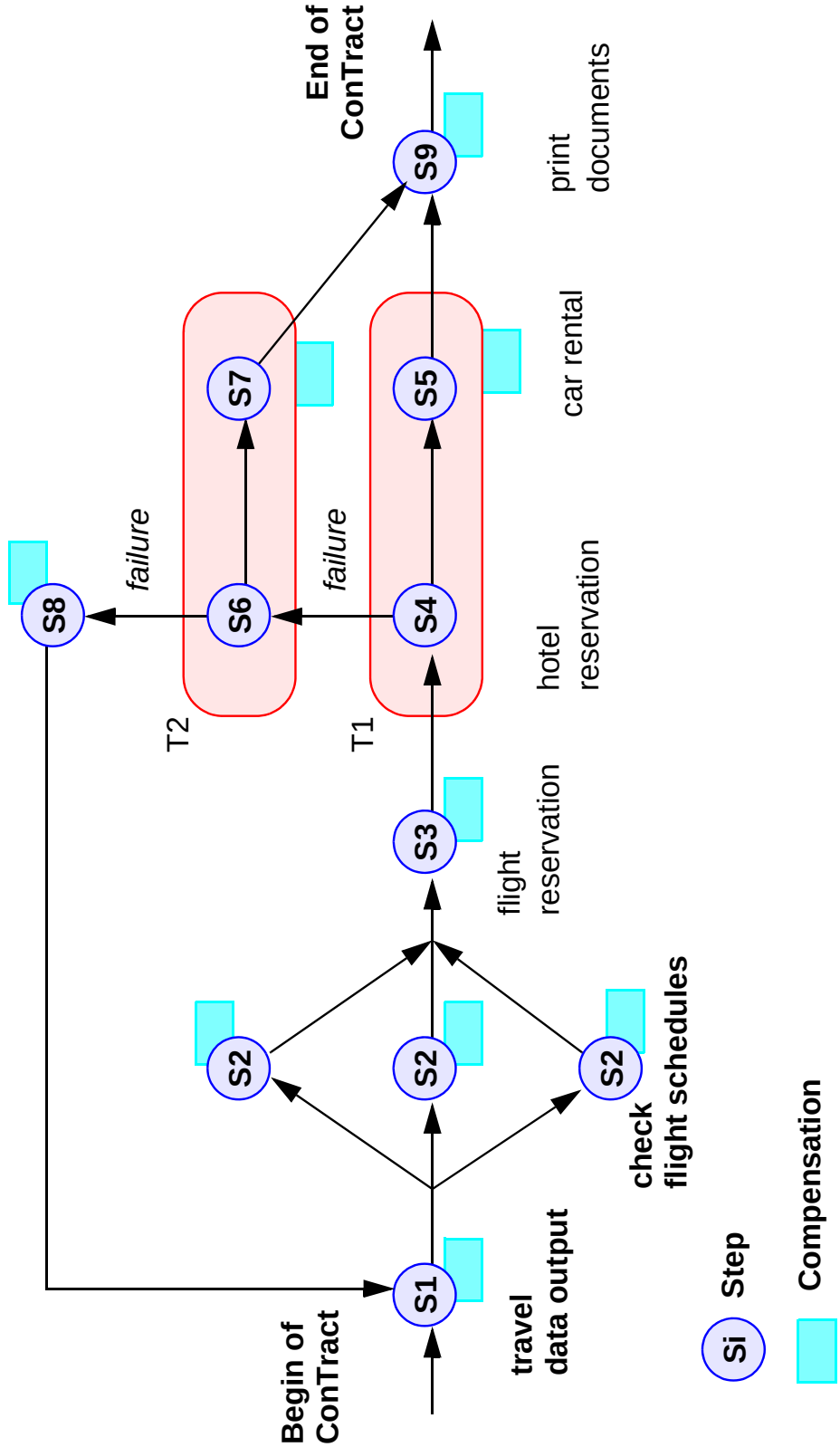
- **ConTracts:[7] Erweiterung des Saga-Ansatzes um**
  - reichere Kontrollstrukturen
    (Sequenz, Verzweigung, Parallelität, Schleife usw.)
  - getrennte Beschreibung von Sub-Transaktionen (**Steps**) und
    Ablaufkontrolle (**Skript**)
  - Verwaltung eines persistenten **Kontextes** für globale Variablen,
    Zwischenergebnisse, Bildschirmausgaben usw.
  - Synchronisation zwischen Steps über Invarianten
  - flexiblere Konflikt-/Fehlerbehandlung

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

7. 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)



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: date, 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

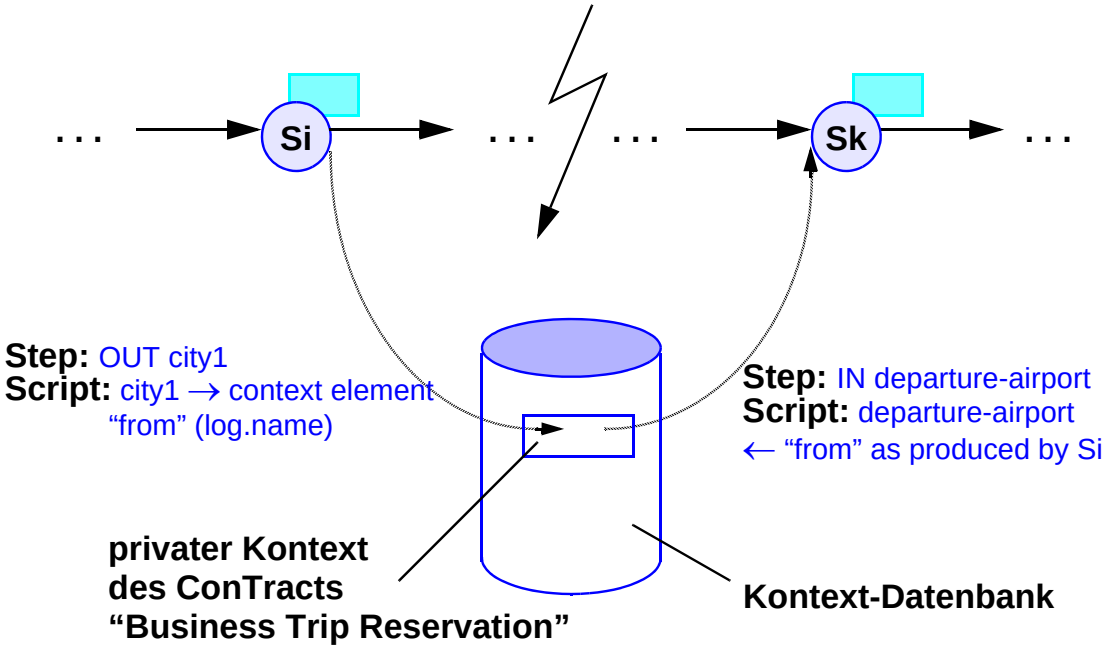
```
T1 (T2, T3 )
```

  - 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)

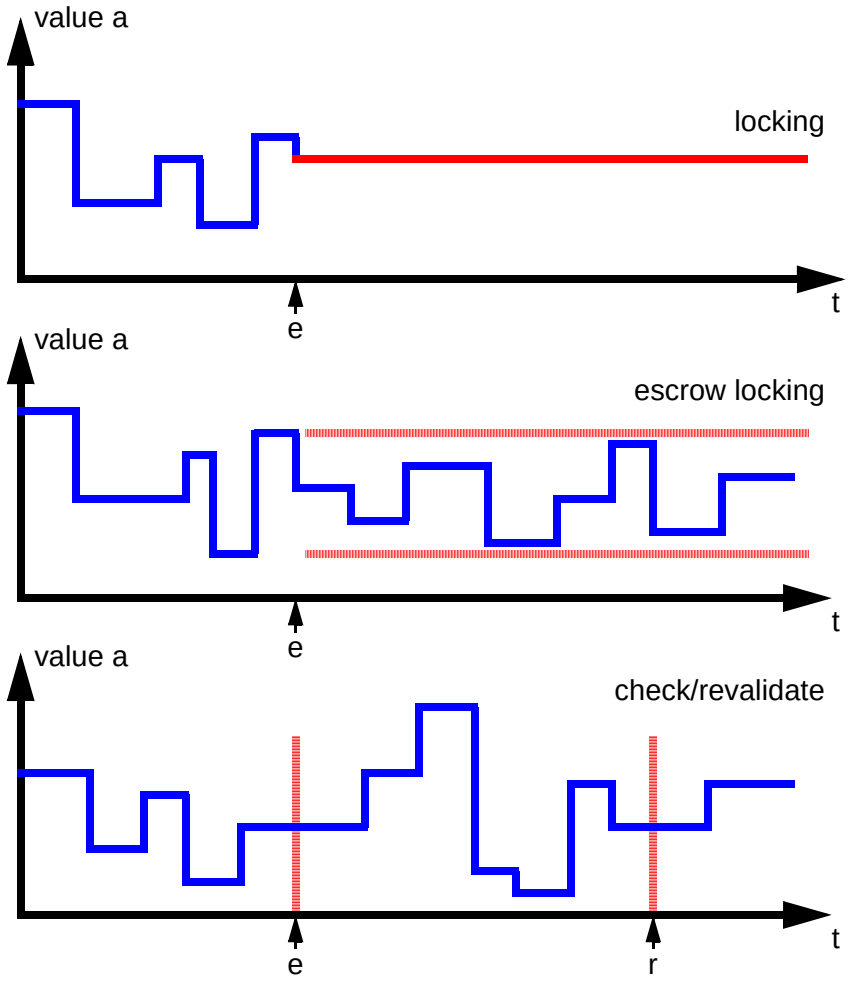
- **Kompensation**
  - Kompensation eines ConTracts ausschließlich auf explizite Benutzeranweisung – nicht automatisch
  - Regeln:
    - für jede(n) Step/Transaktion muß genau eine Kompensations-transaktion vorhanden sein
    - mit dem Commit des Steps müssen alle Daten, die für die Kompensation gebraucht 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

  Â 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

Elektronischer Handel

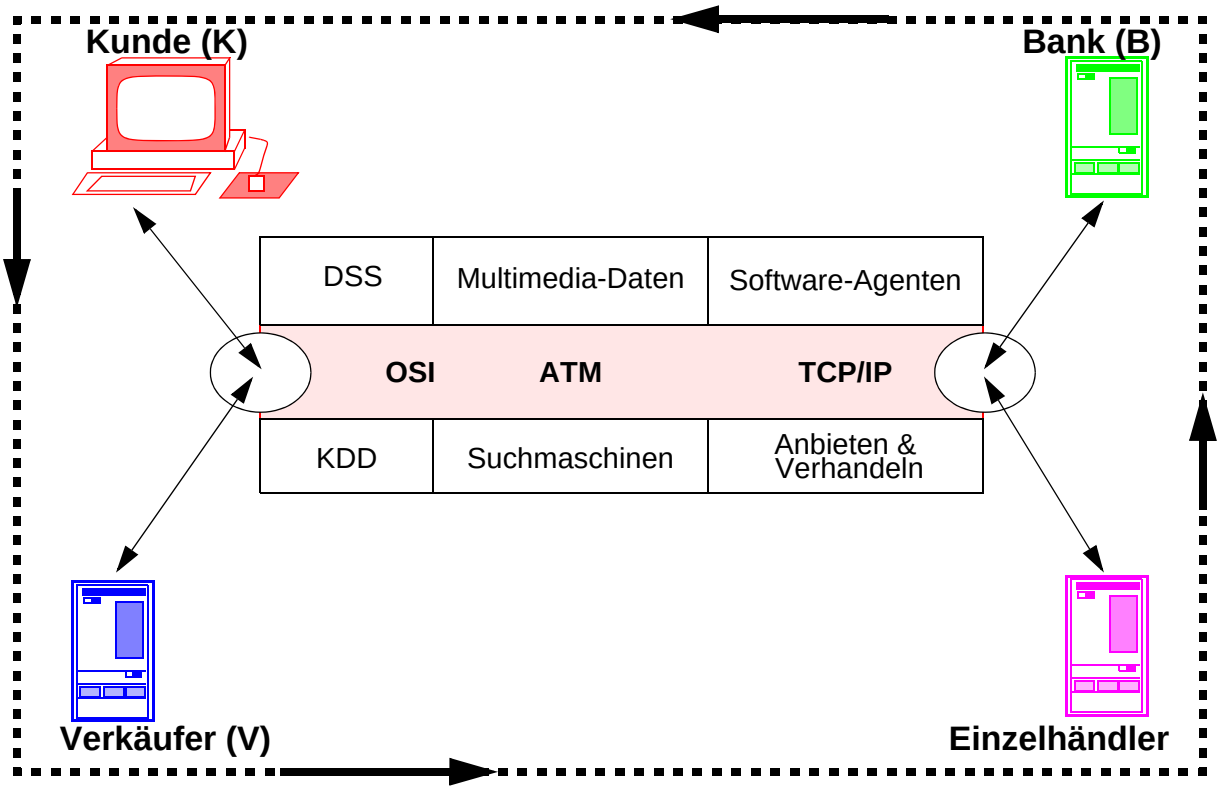
- **Situation**
  - Globalisierung, „Jeder kann jeden erreichen"
  - > 2 * 10^9 Personen im Internet
  - Geschäftstransformation („Virtuelle Firma")
  - „Disaggregation" der Wertschöpfungskette (value chain)

- **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[8]**



8. 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)

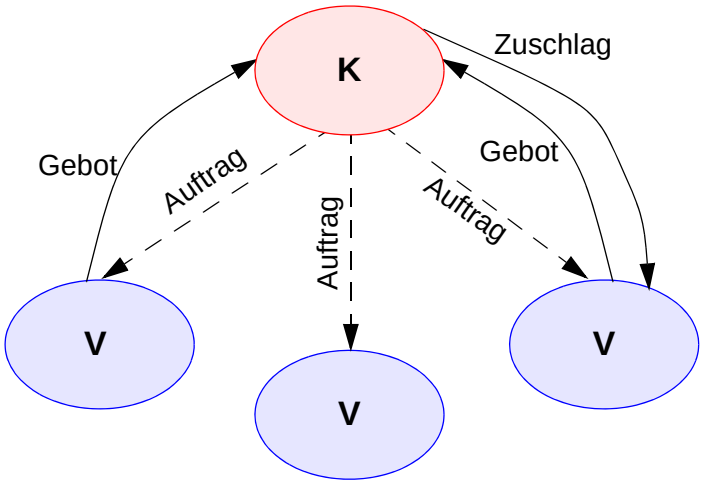
- **Anzahl der Verkäufer: Kunden**
  - 1 : 1 : Einzelgeschäft
  - n : 1 : Angebot
  - 1 : n : Auktion
  - n : m : Börse

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

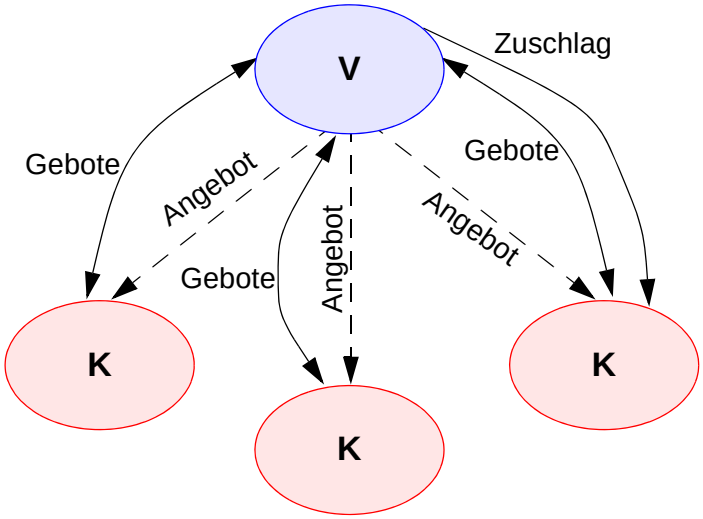
- **Abläufe beim EH**
  - im Internet, verteilte Verarbeitung (zwischen **K**, **V**, **B**)
  - Wie läßt sich Atomarität (und C’ I’ D’)
    bei verteilter Verarbeitung zwischen **K**, **V** und **B** erzielen?

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

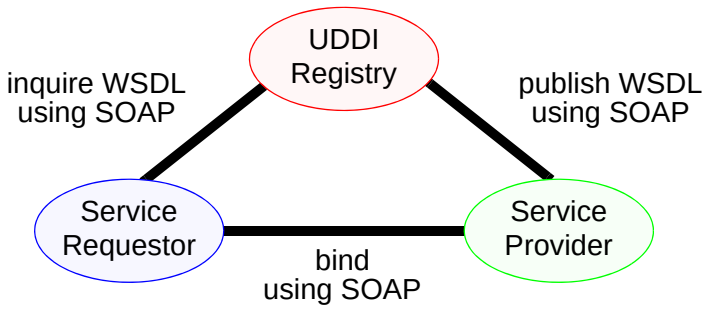


  - **Arten**
    - englische (up), holländische (down) Auktion
    - Vickrey-Auktion (second-price sealed-bit auction)
    - Höchstpreis-Auktion (first-price sealed-bit auction)
  - Formen: C2C-, B2C-, B2B-Auktion

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[9] 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



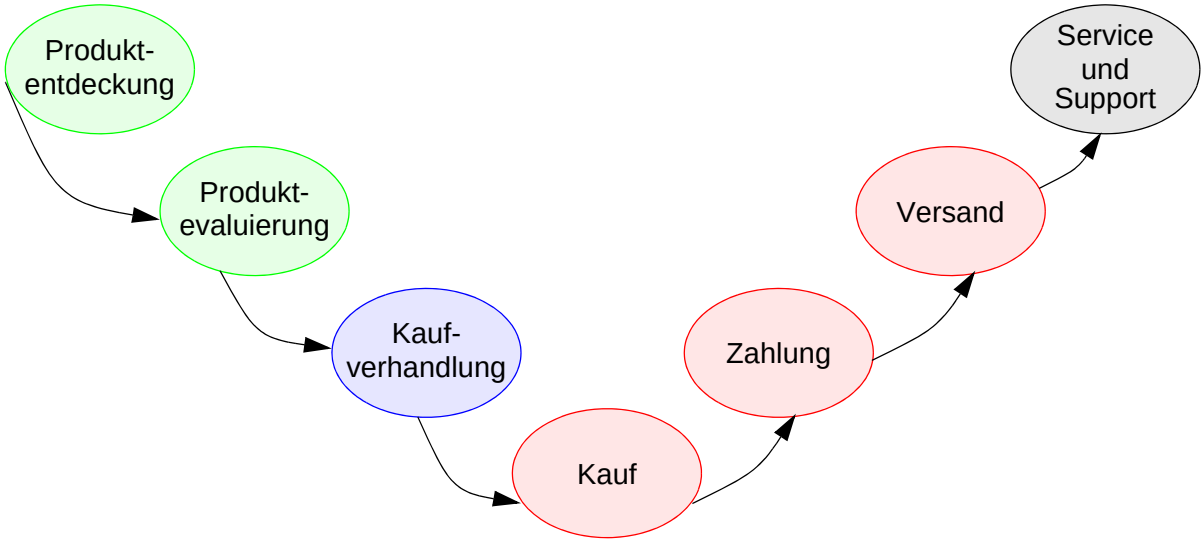
9. A Web service (WS) is a software application identified by a URL, whose interfaces and binding are capable of being defined, described, and discovered by XML artifacts, and supports direct interactions with other software applications using XML-based messages via Internet-based protocols.

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)[10]**
  - sind komplex
  - schließen mehrere (nicht-vertrauenswürdige) Teilnehmer ein
  - überspannen Organisationsgrenzen
  - sind langlebig

- **Geschäftsablauf beim EH (121)**

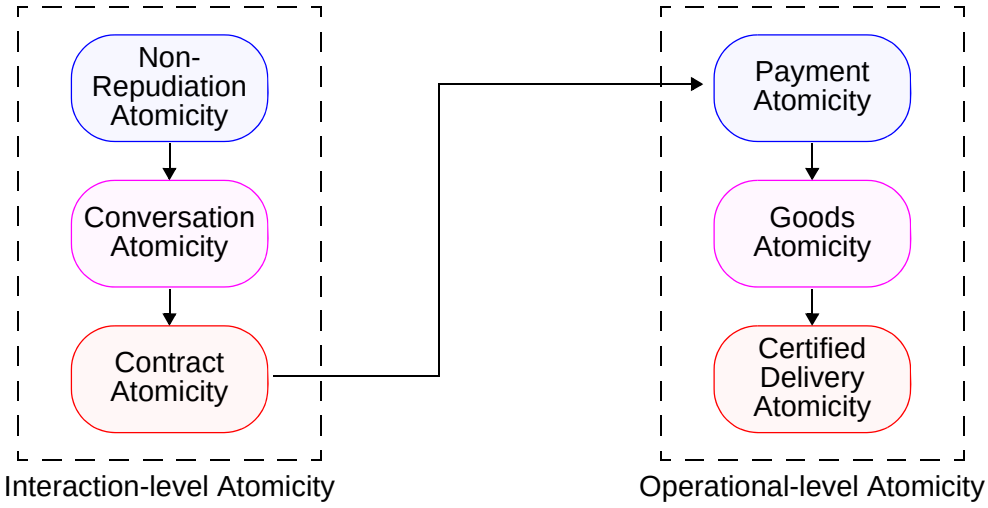


  - automatisierte Geschäftsabläufe mit Verhandlung, Vertragsabschluß, Lieferung mit Tracking, Bezahlung, Ausnahmebehandlung, ...
  - Welche TA-Eigenschaften können gefordert werden?

10. 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[11] 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 Konten-bewegungen sind die Grundelemente von BTs

11. 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

  Â **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)

  Â **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

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

- **Eigenschaften der BT-Atomaritätstypen**
  - Vertragsatomarität Â Konversationsatomarität Â Verbindlichkeitsatomarität
  - Zertifizierte Lieferung Â Warenatomarität Â 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)

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

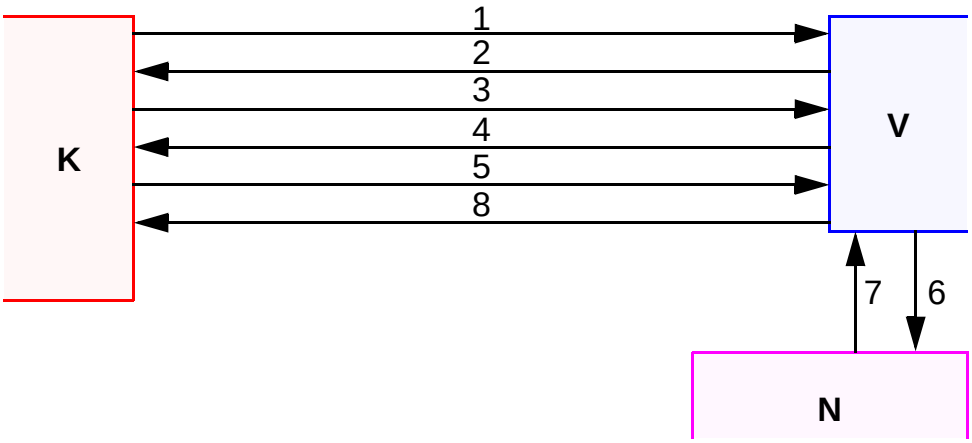
- **EH-Protokolle**
  - Digicash (nicht einmal ZA)
  - First Virtual
  - SSL
  - SET

  Â **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**[12]

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

    Â **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

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

12. 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

  Â **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

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