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 und globale Atomarität
- **Daten- und Funktionsintegration**
  - Ausführung von föderierten Funktionen als Sammlung lokaler Funktionen
  - Komponente zur Funktionsintegration (KIF) der föderierten Funktionen
  - Anwendungsintegration
- **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

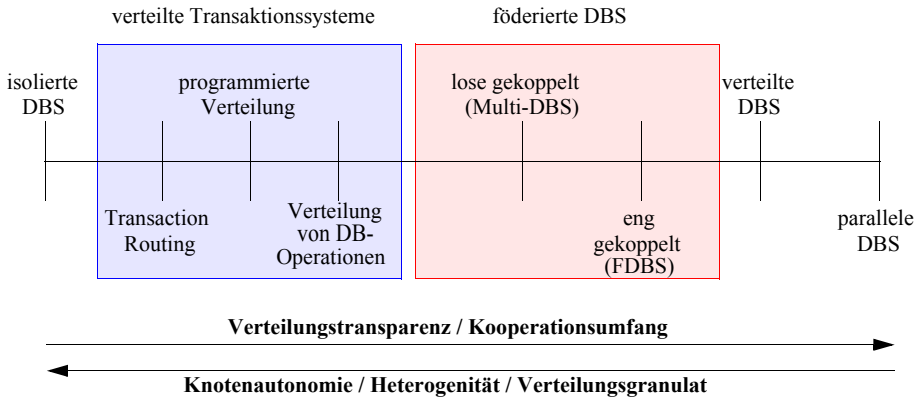
➥ **Web Services[1] zielen auf diese extremste Form der Autonomie ab und verlangen deshalb eine sehr lose Form der Kopplung!**

- **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]**

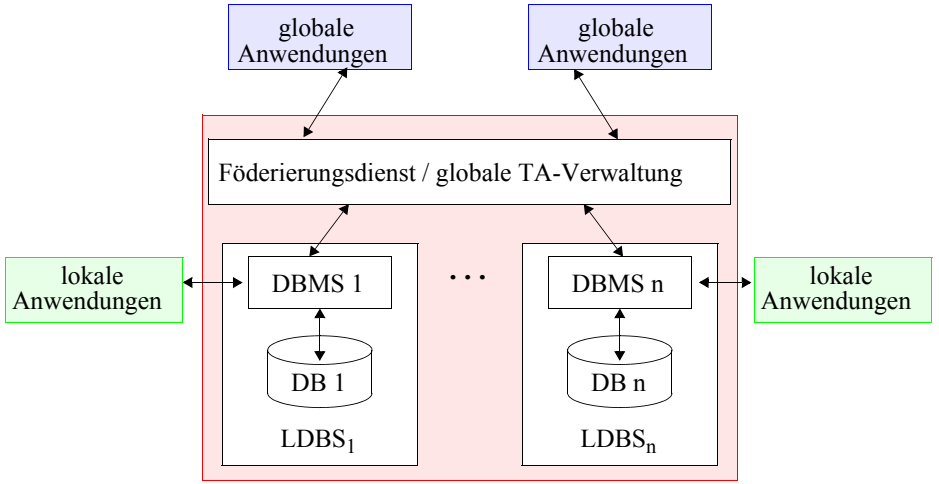


- **Verknüpfung von heterogenen Komponentensystemen durch**
  - **Föderierte DBS (FDBS)**
    - „engere“ Kopplung
    - oft homogenisierte Datensicht (externe Schemata) für globale Anwendungen
    - teilweise Aufgabe der Autonomie
    - Vor- und Nachteile von föderierten Systemen im Allgemeinen
  - **Multi-Datenbanksysteme (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**



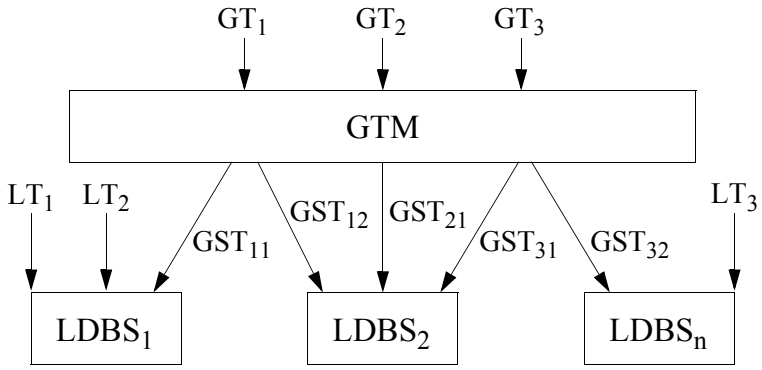
- Autonomie sehr wichtig, da Systeme zum Zeitpunkt der Integration bereits bestehen
- Föderierungsdienst / globale TA-Verwaltung erlauben eine Integration und Nutzung einer Menge von „isolierten“ und heterogenen DBS[3]
  - globale Anwendungen können auf alle Daten der beteiligten DBS zugreifen
  - lokale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

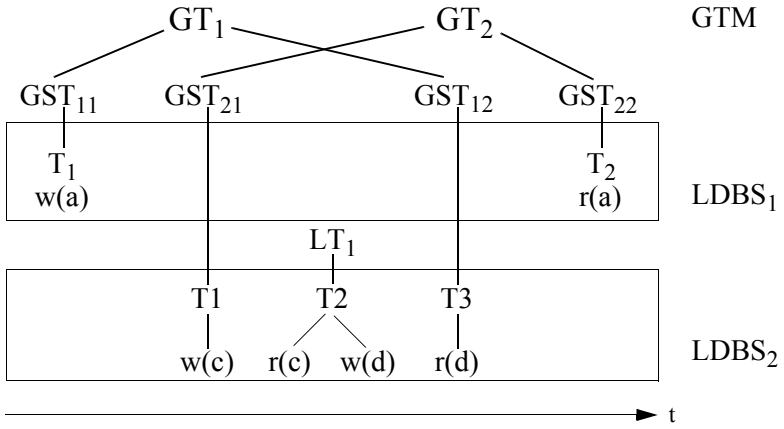
- **Jedes beteiligte DBS (LDBS) ist autonom**
  - kann im Prinzip eigenes Datenmodell besitzen
  - kann eigene Algorithmen für Synchronisation und Recovery einsetzen

➥ **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

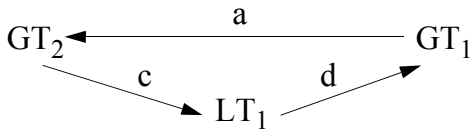
- **Beispiel**



  - 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

  ➥ 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

  ➥ 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

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

  - LDBS entscheiden unabhängig über das Commit lokal ausgeführter TA

  ➥ 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

  ➥ Es ist keine Anpassung der SW (Schnittstellen) erforderlich

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

Probleme heterogener TA-Systeme

- **Globale Serialisierbarkeit**
  - FDBS besteht aus $LDBS_1$ und $LDBS_2$. Objekte a und b werden von $LDBS_1$, c und d von $LDBS_2$ verwaltet. Es werden zwei globale TA ausgeführt:

    GT_1 : $r_1(a)\ r_1(c)$
    GT_2 : $r_2(b)\ r_2(d)$

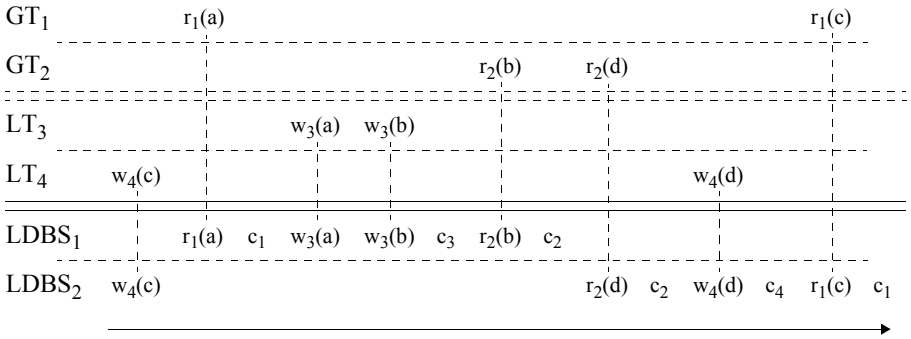
  - Gleichzeitig werden zwei lokale TA ausgeführt:

    LT_3 : $w_3(a)\ w_3(b)$
    LT_4 : $w_4(c)\ w_4(d)$

  - Zwei mögliche lokale Schedules:

    S_1 : $r_1(a)\ c_1\ w_3(a)\ w_3(b)\ c_3\ r_2(b)\ c_2$
    S_2 : $w_4(c)\ r_2(d)\ c_2\ w_4(d)\ c_4\ r_1(c)\ c_1$

  - Graphische Veranschaulichung:



  - lokale Serialisierungsreihenfolgen

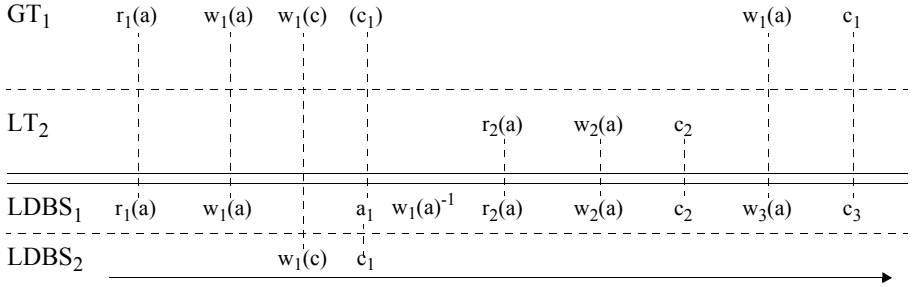
    $LDBS_1$:

    $LDBS_2$:

➥ **nicht auflösbarer Widerspruch auf globaler Ebene**

Probleme heterogener TA-Systeme (2)

- **Globale Atomarität**
  - Sub-TA einer GT müssen entweder alle mit Commit oder alle mit Abort abschließen ➥ setzt „Prepared“-Zustand voraus!
  - lokale Ausführungsautonomie erlaubt beliebigen TA-Abbruch
- **Beispiel:** $LDBS_1$ verwaltet das Objekt a und $LDBS_2$ das Objekt c.
  - Erfolgreiche Ausführung von GT_1: $r_1(a)\ w_1(a)\ w_1(c)$
  - Commit der GT_1 in $LDBS_2$ erfolgreich, in $LDBS_1$ jedoch gescheitert
  - Ausführung von LT_2 in $LDBS_1$: $r_2(a)\ w_2(a)$
  - Abbruch von GT_1 nicht mehr möglich, Wiederholung von $r_1(a)\ w_1(a)$?
  - Erfolgreiche Sub-TA ($r_1(a)$ -> $w_1(c)$) basiert auf altem Wert von a und kann nicht mehr verändert werden
  - Deshalb Versuch der Wiederholung von $w_1(a)$ (als LT_3 mit $w_3(a)$ in $LDBS_1$)
  - Ablauf:



aus Sicht von $LDBS_1$: $r_2(a)\ w_2(a)\ c_2\ w_3(a)\ c_3$ (commit-abgeschlossen)
für GTM Ablauf in $LDBS_1$: $r_1(a)\ r_2(a)\ w_2(a)\ c_2\ w_1(a)\ c_1$

  - Abhängigkeitsfolge:
  - Lösung?

Probleme heterogener TA-Systeme (3)

- **Globale Deadlock-Erkennung und -Vermeidung**
  - 2PL-Protokoll erlaubt Erkennung bzw. Vermeidung von Deadlocks in einem DBS
  - Jedoch keine „Wait-for“-Graphen für gesamtes Komponentensystem
  - Annahme: 2PL in $LDBS_1$ (mit Objekten a und b) und in $LDBS_2$ (mit Objekten c und d)
  - Globale und lokale TA mit

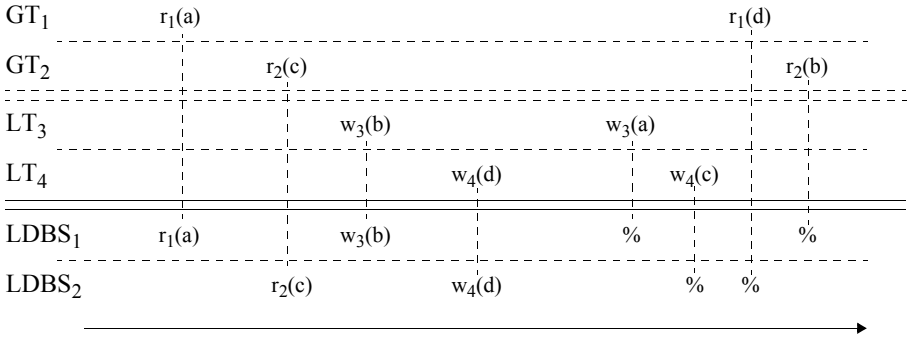
    GT_1 : $r_1(a)\ r_1(d)$

    GT_2 : $r_2(c)\ r_2(b)$

    LT_3 : $w_3(b)\ w_3(a)$

    LT_4 : $w_4(d)\ w_4(c)$

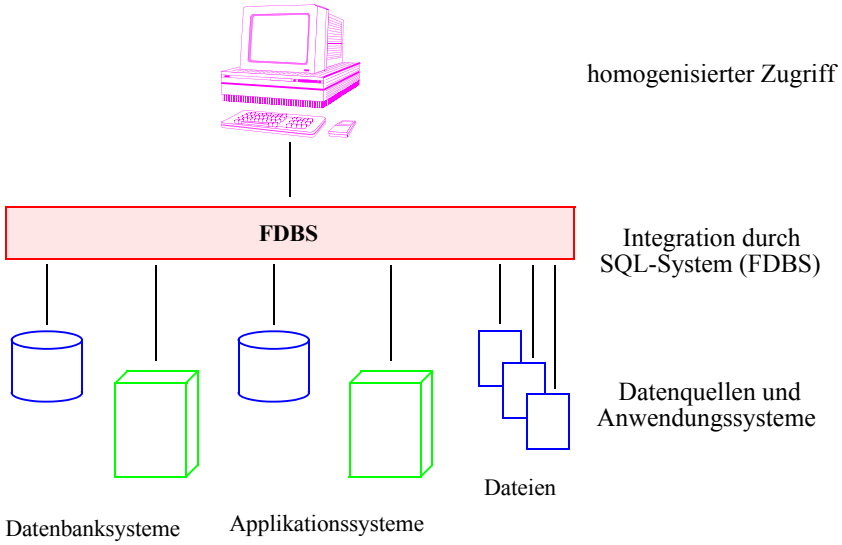
  - Ablauf:



  - Aus jeweils lokaler Sicht von $LDBS_1$ und $LDBS_2$ und aus Sicht von GTM gibt es keine Verklemmung
  - „Blinde“ Deadlock-Auflösung mit Timeout?
  - Jede Lösung erfordert Beschränkung der Autonomie und globale Verfügbarkeit von „lokaler“ Information!

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)



➥ **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

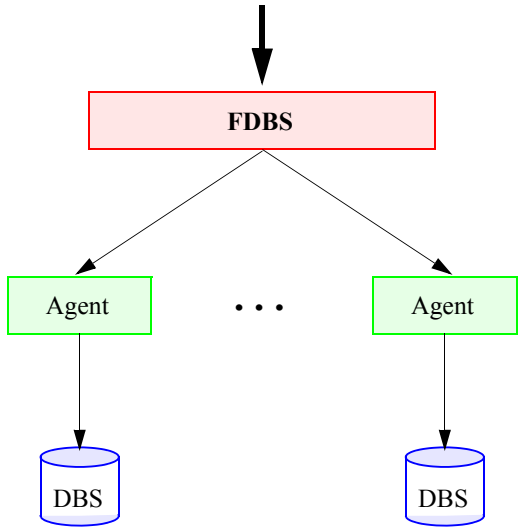
➥ **Einsatz von nicht-wiederherstellbaren Systemen verbietet sich für unternehmenskritische Daten und Anwendungen!**

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

➥ **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

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

Korrektheitsbedingungen bei Mehrebenen-TA

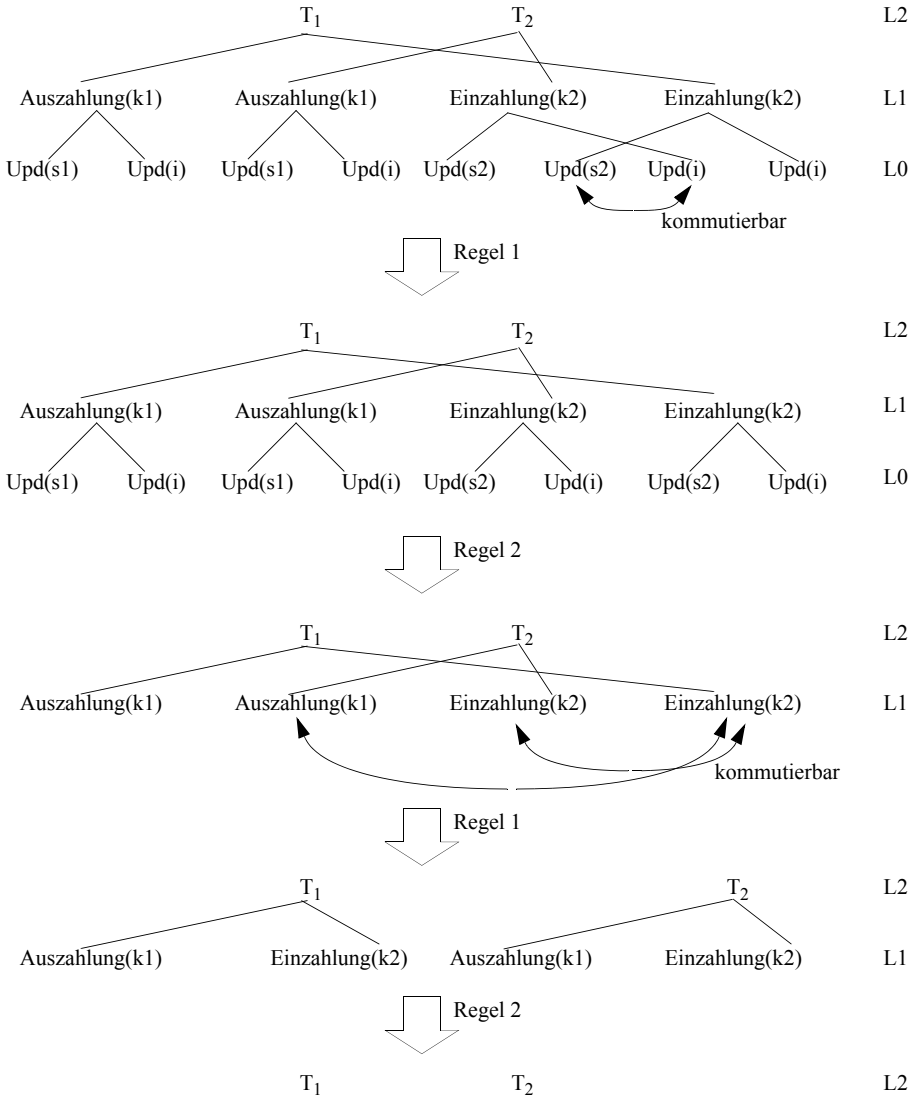
- **Globale Serialisierbarkeit**
  - kann durch Schicht-zu-Schicht-Serialisierbarkeit garantiert werden (level-to-level serializability (L2L))
  - läßt sich überprüfen durch Umwandlung eines serialisierbaren Mehrebenen-Schedule in einen seriellen Schedule

- **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)

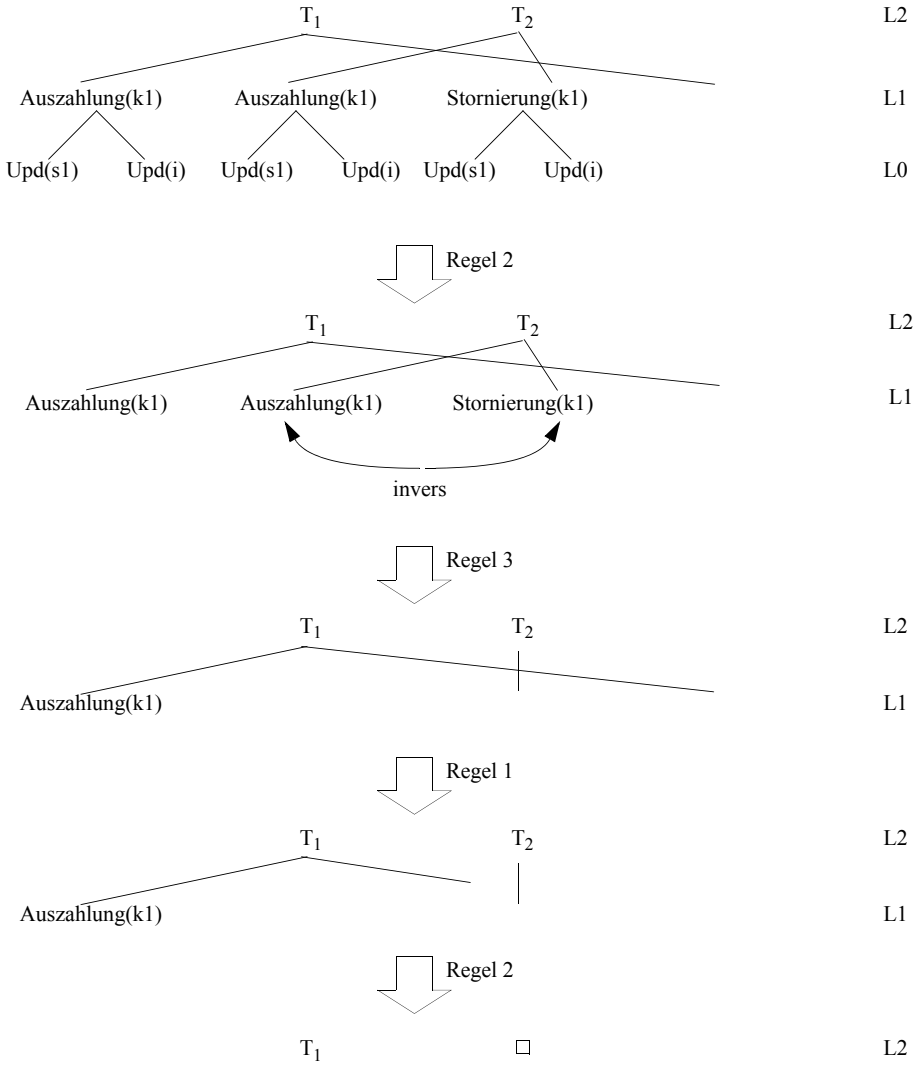
- **Beispiel**



➥ **Schrittweise Konvertierung in einen seriellen Schedule**

Korrektheitsbedingungen bei Mehrebenen-TA (3)

- **Beispiel**
  - *Regel 3: Kompensation von Operationen*

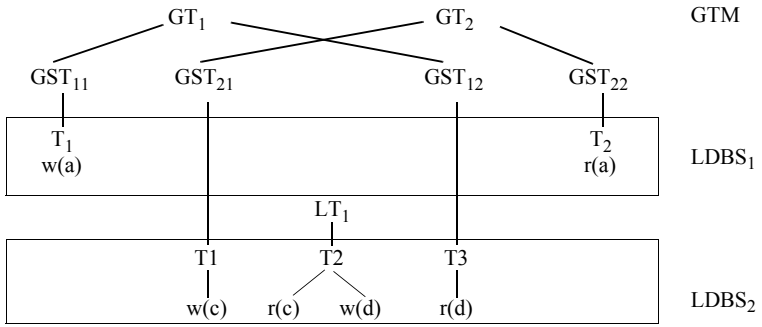


➥ **Behandlung einer abgebrochenen Mehrebenen-TA**

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

TA-Verwaltung – Globale Serialisierbarkeit (2)

- **Früheres Beispiel:**

  GT_1 : $r_1(a)\ r_1(c)$
  GT_2 : $r_2(b)\ r_2(d)$

  LT_3 : $w_3(a)\ w_3(b)$
  LT_4 : $w_4(c)\ w_4(d)$

  - Zwei mögliche lokale Schedules:

    S_1 : $r_1(a)\ c_1\ w_3(a)\ w_3(b)\ c_3\ r_2(b)\ c_2$
    S_2 : $w_4(c)\ r_2(d)\ c_2\ w_4(d)\ c_4\ r_1(c)\ c_1$

- **Einsatz von zurückgehaltenen Sperren (RL)**

  S_1 : $r_1(a)\ c_1$

  S_2 : $w_4(c)\ r_2(d)\ c_2$

  - lokale Serialisierungsreihenfolgen

    $LDBS_1$:

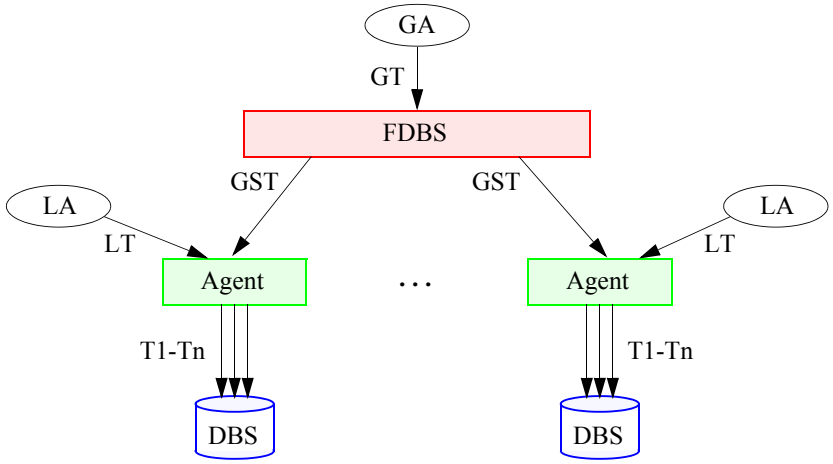
    $LDBS_2$:

- **Erweiterungen**

  erforderlich für Konflikterkennung, da Agent nicht zwischen lokalen TA und globalen Sub-TA unterscheiden kann

TA-Verwaltung – Globale Serialisierbarkeit (3)

- **Anforderungen**
  - Globale Sub-TA müssen identifizierbar sein (z.B. durch zusätzliche Anweisungen BEGIN SUBTRANSACTION und END SUBTRANSACTION)
  - Sub-TA müssen Sperren bis zum Ende ihrer globalen TA zurückhalten
  - Ende der globalen TA muß erkennbar sein
- **Alle TA laufen über die Agenten**



  - Agent muß zwischen lokaler Anwendung und lokalem DBS plaziert werden
  - Agent bietet DBS-Schnittstelle, so daß lokale Anwendungen nicht modifiziert werden müssen
  - Erweiterungen der Schnittstelle durch zusätzliche Operationen möglich

➥ **Agenten bilden zusätzliche Ebene im Mehrebenen-TA-Modell**

  - Vorschlag: Jeder Operationsaufruf einer Agent-TA wird eigene DBS-TA[5]
  - Konsequenzen?

5. Schaad, W.: Transaktionsverwaltung in heterogenen, föderierten Datenbanksystemen, infix, DISDBIS 12, 1996.

TA-Verwaltung – Globale Atomarität

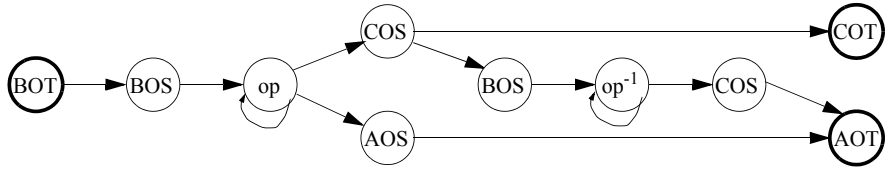
- **Mehrebenen-TA sind offen geschachtelte TA**
  - Sub-TA führen Commit vor globaler TA aus
  - kein Rollback, sondern Kompensation (der SQL-Operationen)
  - aber nur möglich, wenn keine Konflikt-TA dazwischen verarbeitet wurde

➥ **Rolle der zurückgehaltenen Sperren**

- **Was passiert bei Crash eines lokalen DBS?**
  - zurückgehaltene Sperren im Agenten **persistent**, damit Sub-TA (aus der Sicht des lokalen DBS bereits im Commit-Zustand) Crash überleben kann
  - nach Wiederanlauf des lokalen DBS erneute Anforderung der zurückgehaltenen Sperren, bevor neue TA gestartet werden
- **FDBS realisiert 2PC-Protokoll**
  - erste Phase für jede Sub-TA separat
    - Prepared-Zustand der Sub-TA
    - Umwandlung der Sperren in zurückgehaltene Sperren
  - zweite Phase am Ende aller Sub-TA
    - globale TA erreicht Commit
    - alle zurückgehaltenen Sperren werden freigegeben

TA-Verwaltung – Globale Atomarität (2)

- **Zustände einer globalen TA** (für eine Sub-TA gezeigt)



BOT	=	Begin-Of-Transaction	AOT	=	Abort-Of-Transaction
BOS	=	Begin-Of-Subtransaction	AOS	=	Abort-Of-Subtranscation
COT	=	Commit-Of-Transaction	op	=	Operation
COS	=	Commit-Of-Subtransaction	op^{-1}	=	inverse Operation

- **TA-Abbruch einer globalen TA**
  - Rollback der laufenden, Kompensation der erfolgreich beendeten Sub-TA
  - Agent hat alle Operationen, die Kompensation erfordern, in einen Log geschrieben
  - Automatische Ableitung von inversen Operationen durch Agent wünschenswert
    - Kenntnis des DB-Schemas erforderlich!
    - Aufgabe ist auf niederen Sprachebenen einfach (Read/Write)
    - aber bei SQL?
      - IUD-Operationen
      - gespeicherte Prozeduren, Sichten, Regeln und Trigger?

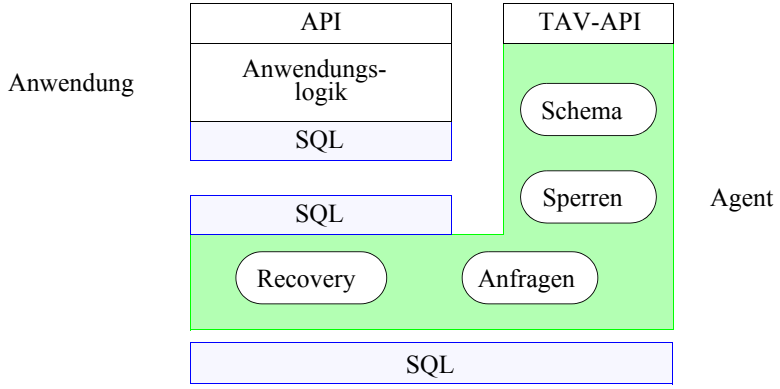
➥ **Automatisierung ist abhängig vom Datenmodell**

- **Globale Deadlock-Erkennung**
  - wenn verfügbar, Nutzung von Wait-for-Information aus lokalen DBS
    - betrifft lokale TA und Sub-TA
    - Kommunikation zwischen Agenten und FDBS
  - sonst: Timeout-Verfahren

Rolle der Agenten

- **Aufgaben des Agenten**
  - vor allem: Konflikterkennung und -verhütung zwischen lokalen TA und globalen Sub-TA
  - FDBS ist aus Sicht des Agenten eine Anwendung
    - spezialisierte Schnittstelle für Koordination zum Erreichen globaler Serialisierbarkeit, globaler Atomarität usw.
    - Austausch von Informationen zur effizienteren globalen Deadlock-Erkennung

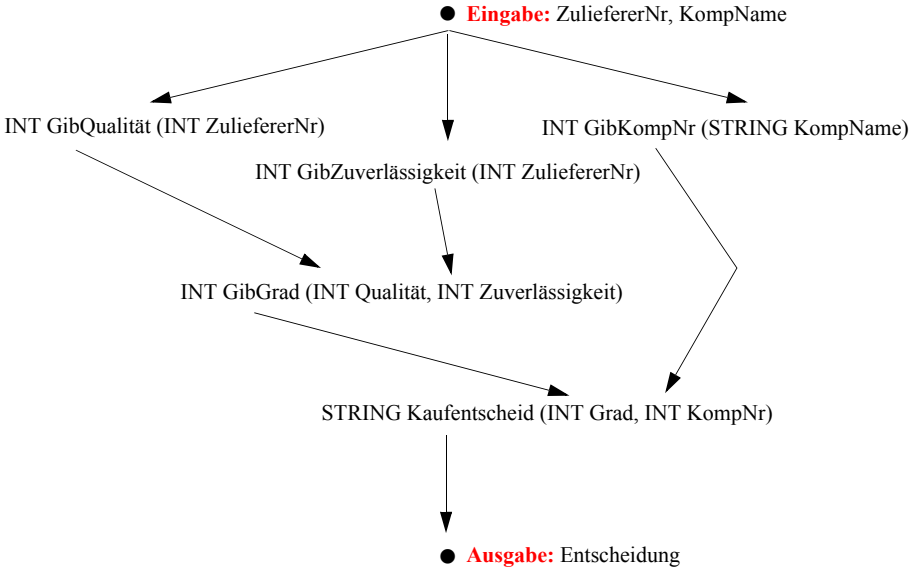
- **Aufbau des Agenten**
  - Anfrageanalysator
  - Schemaverwaltung (Anpassung von Operationen an SQL-Dialekte, Ableitung von Inversen)
  - TA-Verwaltung mit Komponenten für Concurrency Control, Logging und Recovery
  - Komponenten und Schnittstellen



➥ **FDBS-Architektur mit Agenten bietet bisher nur Datenintegration, ist aber schon komplex genug**

Daten- und Funktionsintegration[6]

- **Bisher „manuelle“ Integration mehrerer AWS erforderlich**
  - Scenario: Einkaufsabteilung eines Unternehmens
    - Mitarbeiter wird durch Funktion Kaufentscheid unterstützt
    - Parameter hängen vom Qualitäts- und Zuverlässigkeitsgrad und der Nummer der Komponente ab
    - Einsatz von Funktionen verschiedener AWS erforderlich (GibQualität von Lagerhaltungssystem, GibZuverlässigkeit von Einkaufssystem, GibKompNr von PDMS)

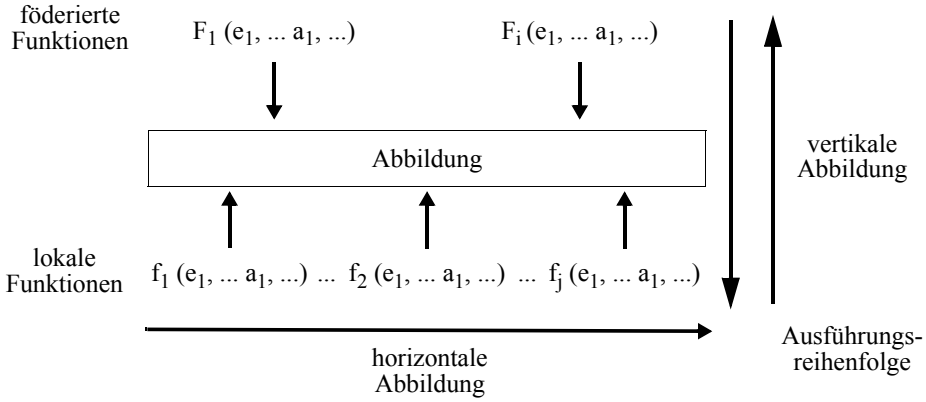


➥ **Ablauf der einzelnen Schritte des Benutzers bei „manueller“ Integration**

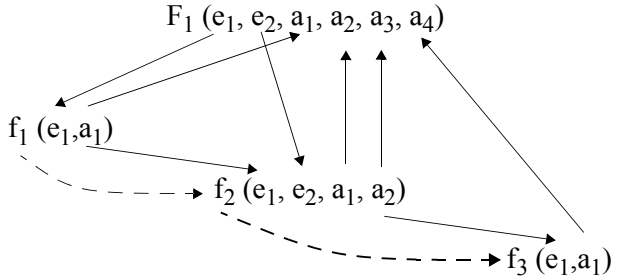
6. **Informationsintegration** wird manchmal als die Verbindung von Daten- und Funktionsintegration angesehen. **Funktionsintegration** bezeichnet das Verfügbarmachen lokaler Funktionen bzw. Dienste aus den einzelnen Systemen in einer einheitlichen Form.

Daten- und Funktionsintegration (2)

- **Automatisierung der Funktionsintegration**
  - Abbildungsmodell: Immer wiederkehrende Sequenzen von (lokalen) Funktionsaufrufen sollen hinter einer föderierten Funktion verborgen werden
  - Ausgangslage



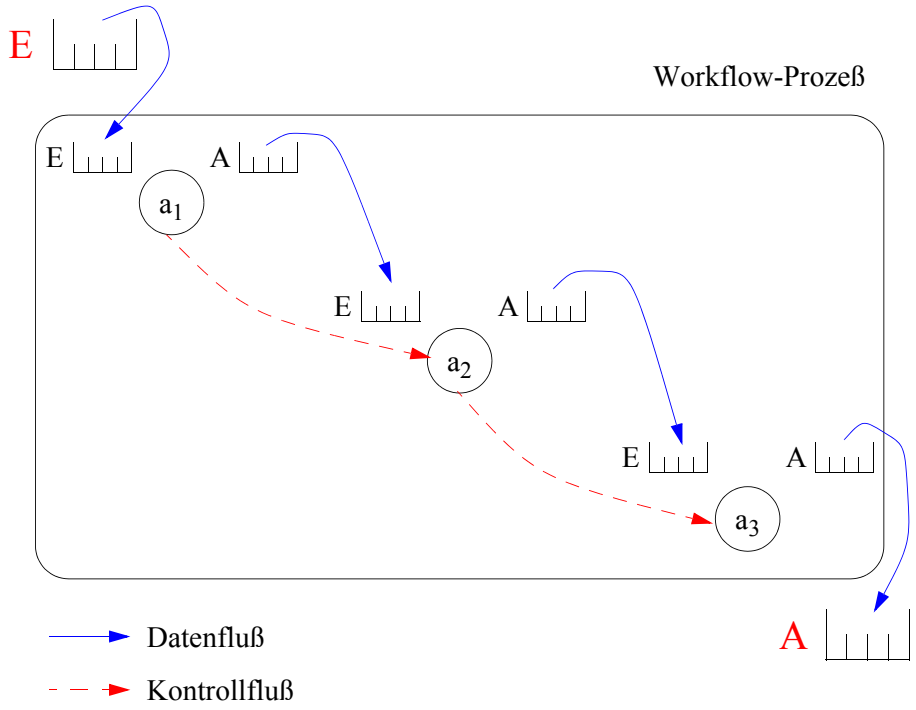
  - Beschreibung durch Abhängigkeiten



➥ **gerichteter, azyklischer Graph: topologische Sortierung!**

Daten- und Funktionsintegration (3)

- **Prozeßmodell eines Workflows**



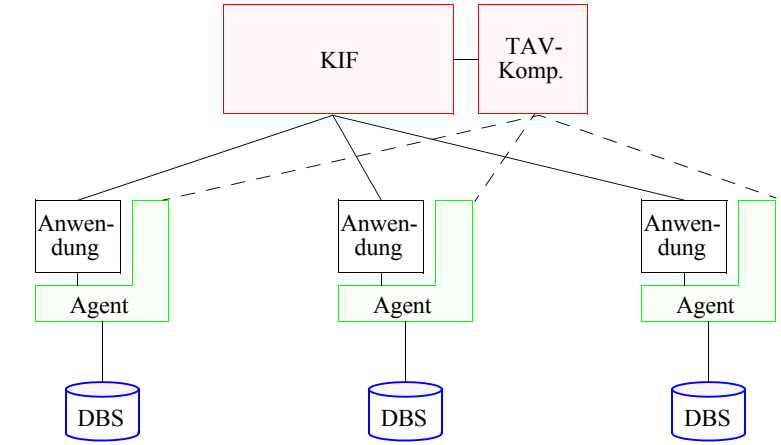
- Übereinstimmungen
  - gerichteter, azyklischer Graph
  - Ausführungsreihenfolge ≡ Kontrollfluß
  - Abhängigkeiten ≡ Datenfluß
  - Funktionen ≡ Aktivitäten

Daten- und Funktionsintegration (4)

- **Wie läßt sich Funktions-/Anwendungsintegration erreichen?**
  - Anwendungssysteme erlauben nur Funktionsaufrufe
  - Mehrere lokale Funktionen können zu einer föderierten Funktion zusammengefaßt werden
  - Aufrufstruktur für Anwendungssysteme, die lokale Funktionen bereitstellen, läßt sich als Graph spezifizieren
  - Komponente zur Funktionsintegration (KIF) kann Graph abarbeiten und Ergebnis als **„abstrakte Tabelle“** zum FDBS übermitteln
  - Zugriff auf föderierte Funktion (über FDBS)

    Select Entscheidung
    From KaufeKomponente
    Where KompName = ‘xyz’ And ZuliefererNr = 1234

  - gleiche TA-Problematik wie bei FDBS

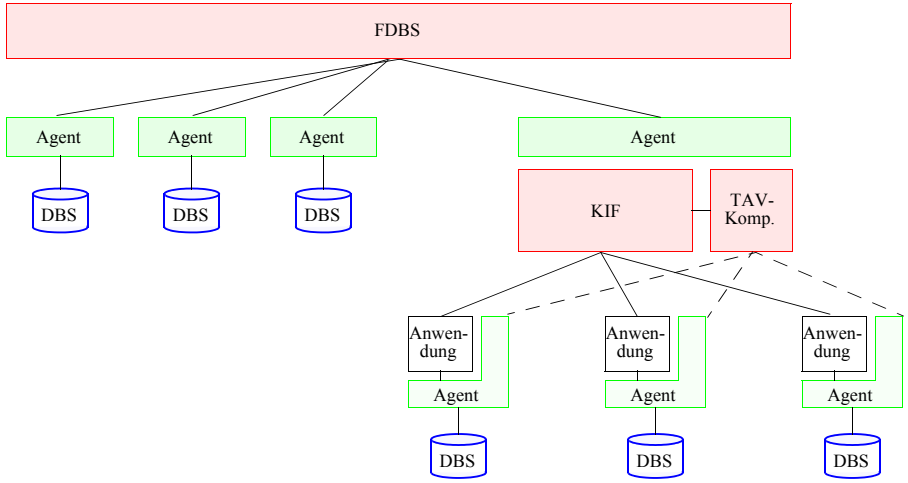


Kommunikation über Funktionsaufrufe zwischen KIF und Anwendungssystem

Kommunikation über TAV-API zwischen TAV-Komponente und Agent

Daten- und Funktionsintegration (5)

- **Vollständige Architektur zur Daten- und Funktionsintegration**



- **Optionen zur Implementierung der KIF**[7]
  - UDTFs (user-defined table functions): eingeschränkte Aufrufstrukturen (z.B. keine Iteration)
  - Flow-System (keine Einschränkungen)
    - WfMS (schwerfällig, Benutzerinteraktion nicht erforderlich)
    - Microflows (sogen. Production Workflows ohne Benutzerinteraktion)
  - J2EE, Web Services, ...

  ➥ **wird nicht vertieft**

- **Anbindung an FDBS** (hier: abstrakt durch Agent)
  - SQL/MED-Wrapper als künftiger Standard
  - UDTFs

  ➥ **wird nicht vertieft**

7. Härder, T., Hergula, K.: Ankopplung heterogener Anwendungssysteme an Föderierte Datenbanksysteme durch Funktionsintegration, in: Informatik – Forschung und Entwicklung 17:3, Sept. 2002, pp. 135-148.

Integration von Anwendungen

- **Stufen der Integration**
  - Die meisten Funktionen eines Unternehmens werden durch SW unterstützt

  ➥ Geschäftsprozess-Integration,
  um weitere Automatisierung und Optimierung der Abläufe zu erreichen

  - Interaktion zwischen mehreren AW-Systemen ist zunehmend auch **unternehmensübergreifend** erforderlich

  ➥ Enterprise Application Integration (EAI),
  z. B. zwischen CRM-System (Customer Relationship Management) des Lieferanten und e-Procurement-System (Beschaffung) des Kunden

  - Schwierigere Integrationsanforderungen, wenn **Daten aus verschiedenen SW-Systemen** verknüpft werden müssen

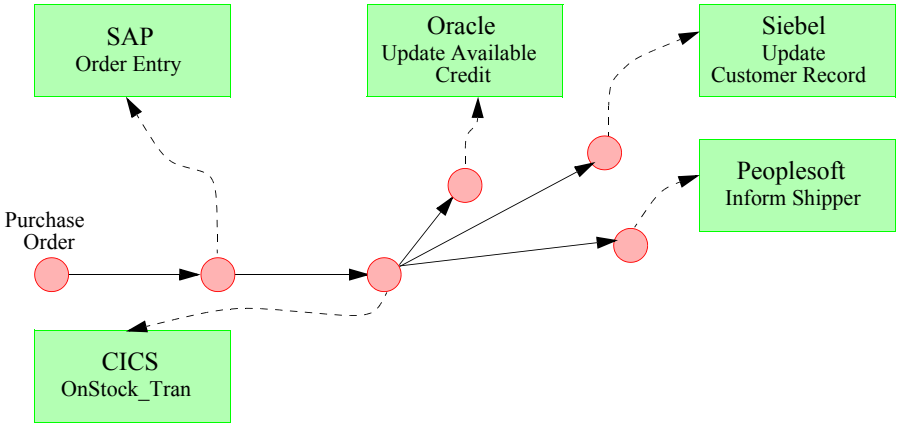
  ➥ Enterprise Information Integration (EII)
  verlangt Behandlung semantischer Aspekte

- **Typische Probleme der Integration**
  - Plattformabhängigkeit
    - Systeme sind oft auf Betriebssystemplattform zugeschnitten
    - keine Interaktionsmöglichkeiten mit anderen Systemen (-> Adapter)
  - Datenformatabhängigkeit
    - unterschiedliche Datenformate und Datenmodelle
  - Prozessabhängigkeit
    - Ablauf, der Interaktion mehrerer Komponenten erfordert, kann oft nur ganz oder gar nicht verändert werden
    - Fehlerbehandlung ist eine Herausforderung
  - Management und Optimierung nach Integration
    - Beurteilung der Sicherheit, Verfügbarkeit und Leistungsfähigkeit des Gesamtsystems ist schwierig: keine Garantie für Ausfall- und Antwortzeiten?
    - Optimierung (Lastbalancierung, Caching) ist schwierig

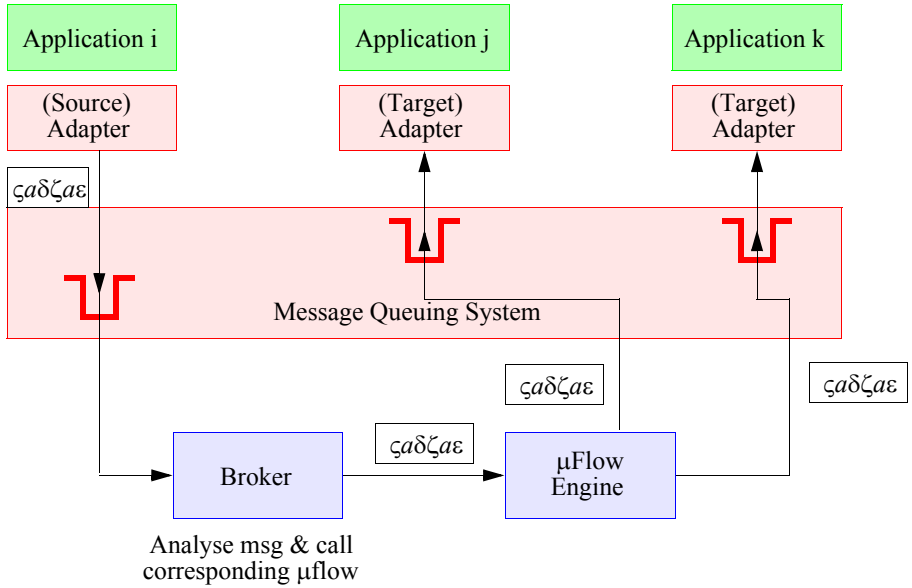
  ➥ Neue Techniken: Autonomic Computing, Grid Computing usw.

Integration von Anwendungen (2)

- **Problemstellung der EAI**
  - heterogene Systeme
  - unternehmensübergreifende Interaktion (Transaktionsschutz)

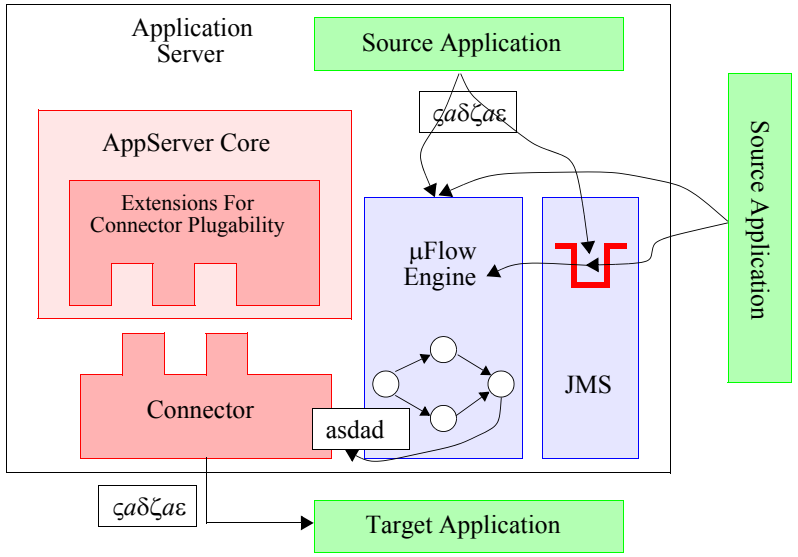


- **Flows & Adapters: „Traditionelle“ EAI**

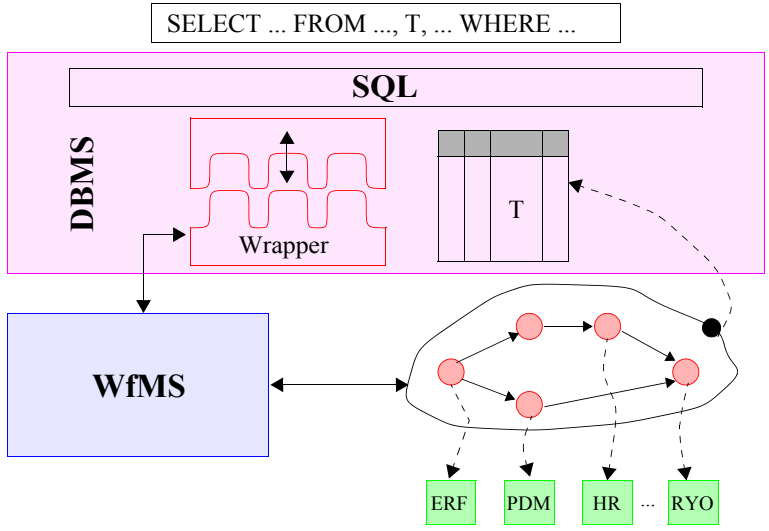


Integration von Anwendungen (3)

- **Flows & Connectors: „Moderne“ EAI**

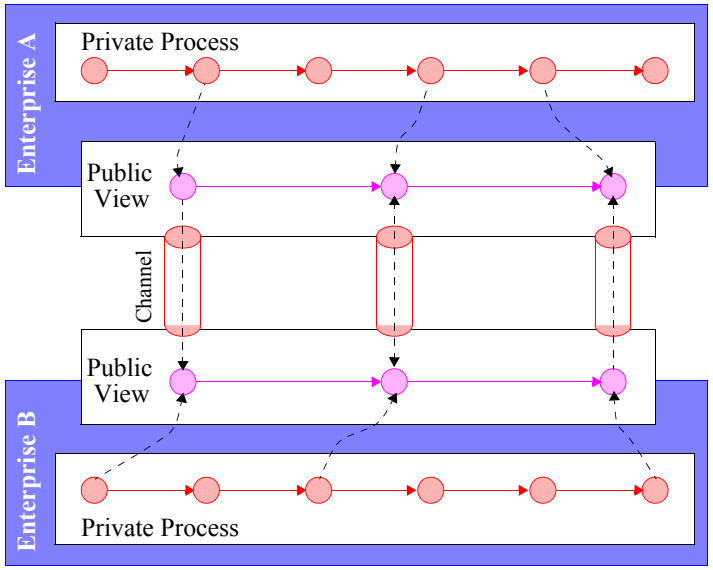


- **Flows & Wrappers: Funktionsintegration mit Datenbanksystemen**

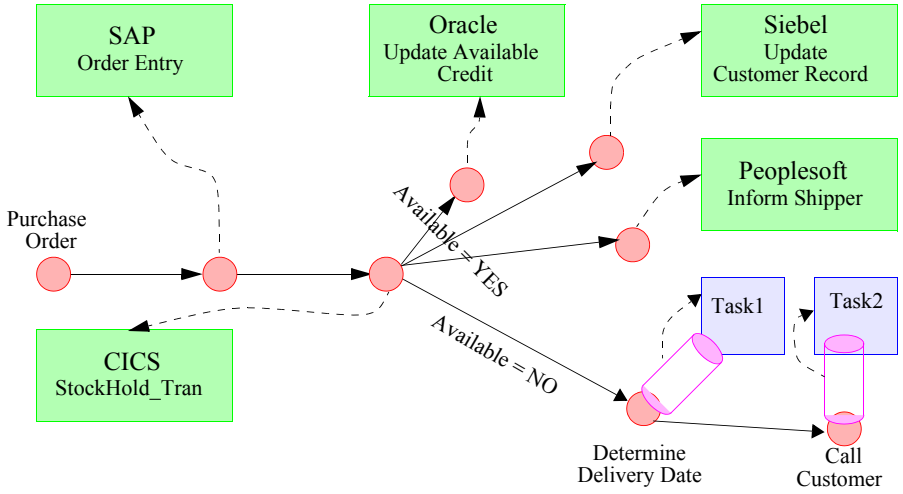


Integration von Anwendungen (4)

- **Flows & Channels: B2B-Interaktionen**



- **Flows & People: Exception Handling**



Integration von Anwendungen (5)

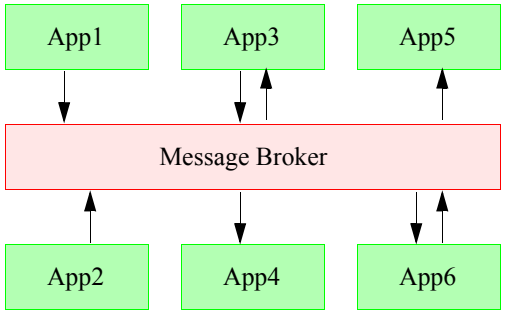
- **Diese EAI-Technologie ist sehr verwirrend**
  - Adapter
  - Connector
  - Wrapper
  - Channel
  - ...

- **Neuer Lösungsansatz nach folgenden vier Prinzipien**
  - Lose Kopplung
    - Systeme bleiben autonom und kommunizieren über Nachrichten
    - gemeinsames Kompilieren und Binden findet nicht statt: unabhängige Weiterentwicklung
  - Virtualisierung
    - Austauschbarkeit von Komponenten
    - Kommunikationspartner werden häufig dynamisch bestimmt
  - Einheitliche Konventionen
    - Dienste unterstützen viele unterschiedliche Datenformate, protokolle und Mechanismen
    - verwendete Technologien werden nicht eingeschränkt
  - Standards
    - Erfolgsrezept ist Festlegung von Standards
    - alle großen Plattform-Anbieter (wie BEA, IBM, Microsoft, ...), Anbieter von Softwareanwendungen (wie SAP, Siebel, ...) und Anwender halten sich daran!

➥ **Sprachansatz: BPEL4WS**
(Business Process Execution Language for Web Services)

Integration von Anwendungen (6)

- **Umsetzung durch Middleware-Architektur für EAI**



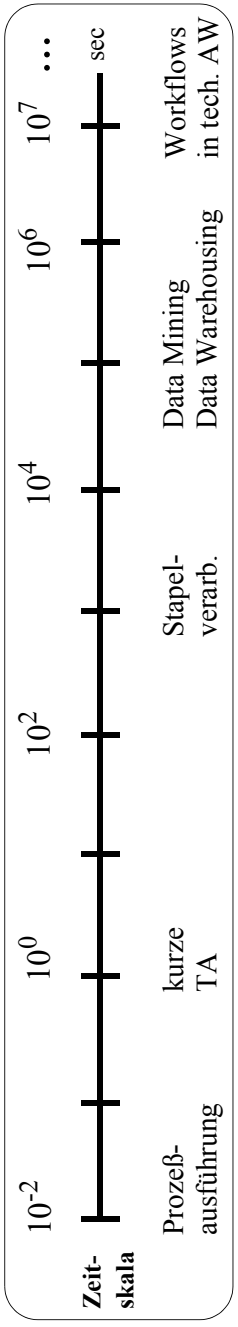
- **Message Broker**
  - leistet eine logische Zentralisierung der Integrationsaufgaben
  - setzt die geforderten vier Prinzipien um
    - lose Kopplung wird erreicht, indem Anwendungen sich nicht direkt aufrufen
    - Virtualisierung wird durch Regelmenge (zur Weiterleitung, Transformation und Protokollierung von Nachrichten) im Message Broker erreicht
    - einheitliche Konventionen und
    - Standards werden durch Message Broker erzwungen

- **Neue Technologie: Web Services**[8]
  - Web Service wird durch einen URI (Uniform Resource Identifier) identifiziert
  - Schnittstelle eines Web Service ist maschinenlesbar und wird durch WSDL (Web Service Definition Language) beschrieben
  - Web Service kommuniziert mit anderen SW-Komponenten durch XML-Nachrichten insbesondere mit Hilfe von Internet-Protokollen (HTTP, SMTP)
  - Web Services sind autonom. Man kann nicht beeinflussen, ob und wie eine Nachricht vom Web Service verarbeitet wird. Qualitätseigenschaften wie Antwortzeitgarantien müssen durch zusätzliche Vereinbarungen geregelt werden

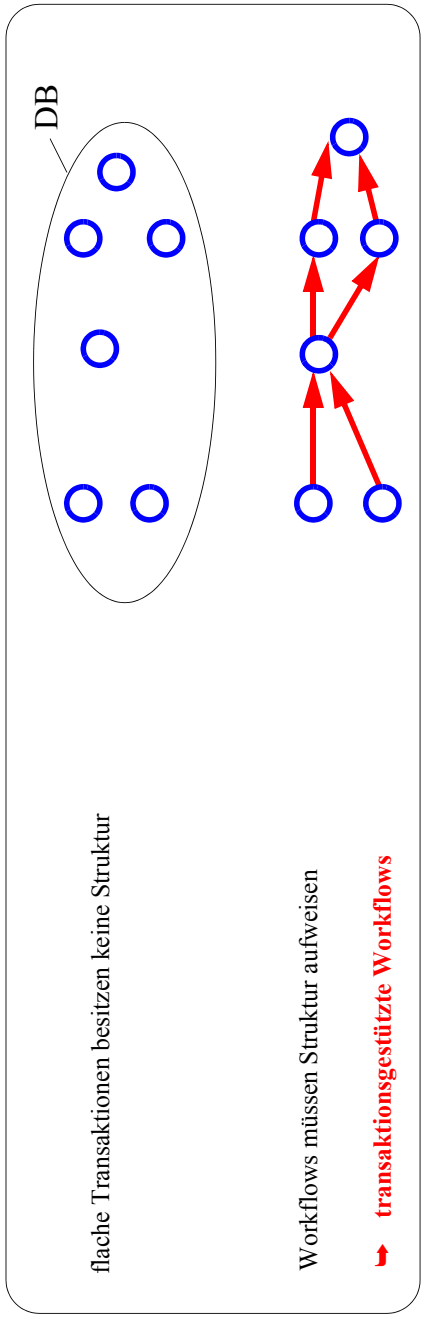
8. W3C-Definition: A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web Service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Workflows

- **Neue Herausforderung – Dauer der Ablaufkontrolle**



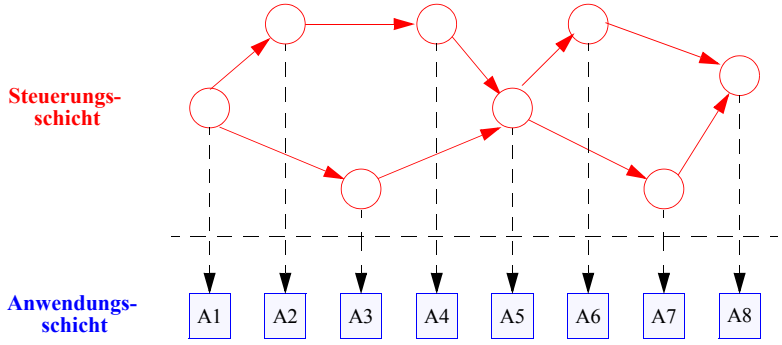
- **Strukturierung der Abläufe: Abhängigkeiten auf gemeinsamen Daten**



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[9]
  - 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)



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

Ausführung von Workflows (2)

- **Essentielle Aspekte von Workflows**

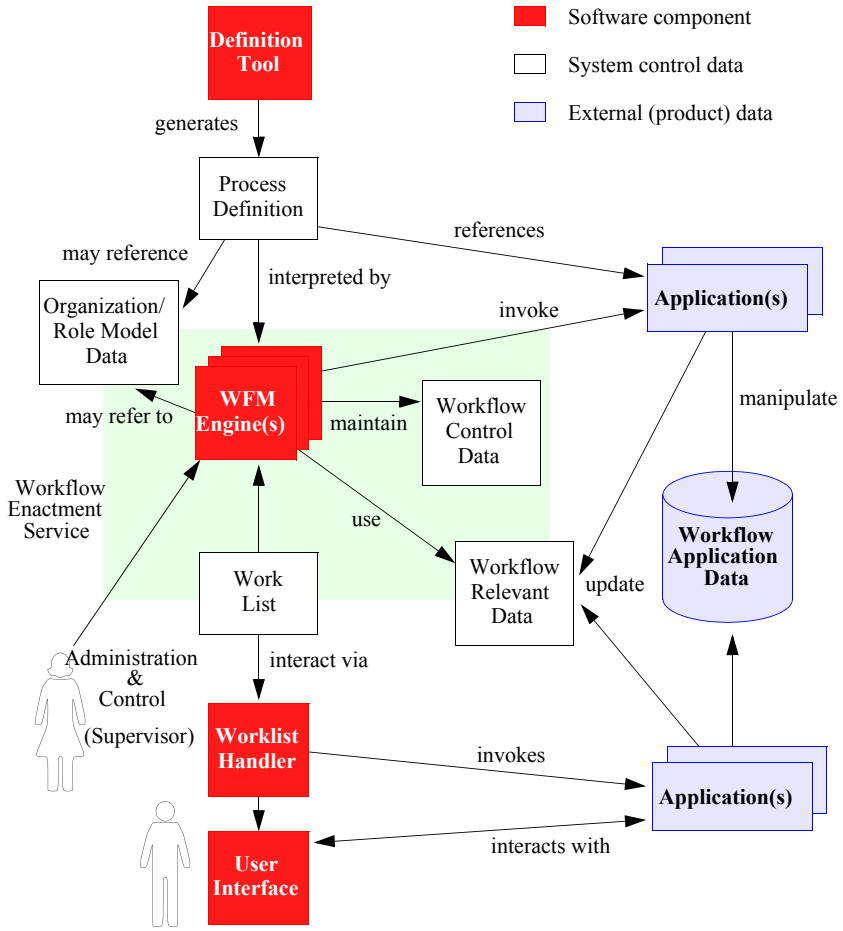
  eingeführt in MOBILE[10]: Forschungsprototyp eines WfMS
  - orthogonale Aspekte
    - ***Funktionaler Aspekt:***
      Was soll ausgeführt werden?
    - ***Datenbezogener Aspekt:***
      Welche Daten werden von Workflows/Workflow-Applikationen konsumiert und produziert?
    - ***Verhaltensbezogener Aspekt:***
      Wann sind Workflows/Workflow-Applikationen auszuführen?
    - ***Operationaler Aspekt:***
      Wie ist eine Workflow-Operation/-Applikation implementiert?
    - ***Organisatorischer Aspekt:***
      Wer hat eine(n) Workflow/-Applikation auszuführen?
    - **Sicherheitsaspekt:**
      Wer ist zur Wf-Ausführung berechtigt?
      (Authentifikation und Autorisierung)
  - weitere technische Aspekte
    - ***transaktionaler Aspekt:***
      Wie wird erreicht, daß Aktivitäten und logisch zusammengehörige Folgen von Aktivitäten atomar und persistent ausgeführt werden?
    - ***historischer Aspekt:***
      Welche Information existiert zu bereits abgeschlossenen Workflow-Ausführungen? (Nachweis, Optimierung)

10. Jablonski, S.; Bussler, C.: Workflow-Management – Modeling Concepts, Architecture and Implementation, Thomson International, 1996.

Ausführung von Workflows (3)

- **Überblick über Aufgaben und Abhängigkeiten**

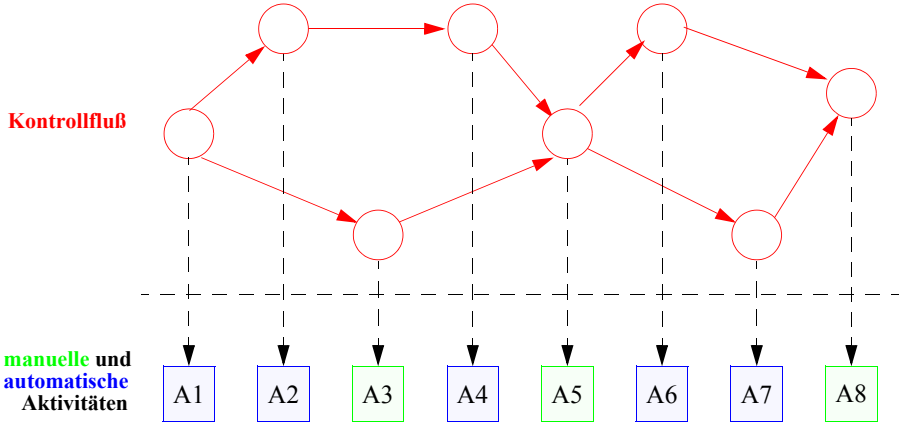
  eines Workflow-Management-System nach WfMC[11]



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

Ausführung von Workflows (4)

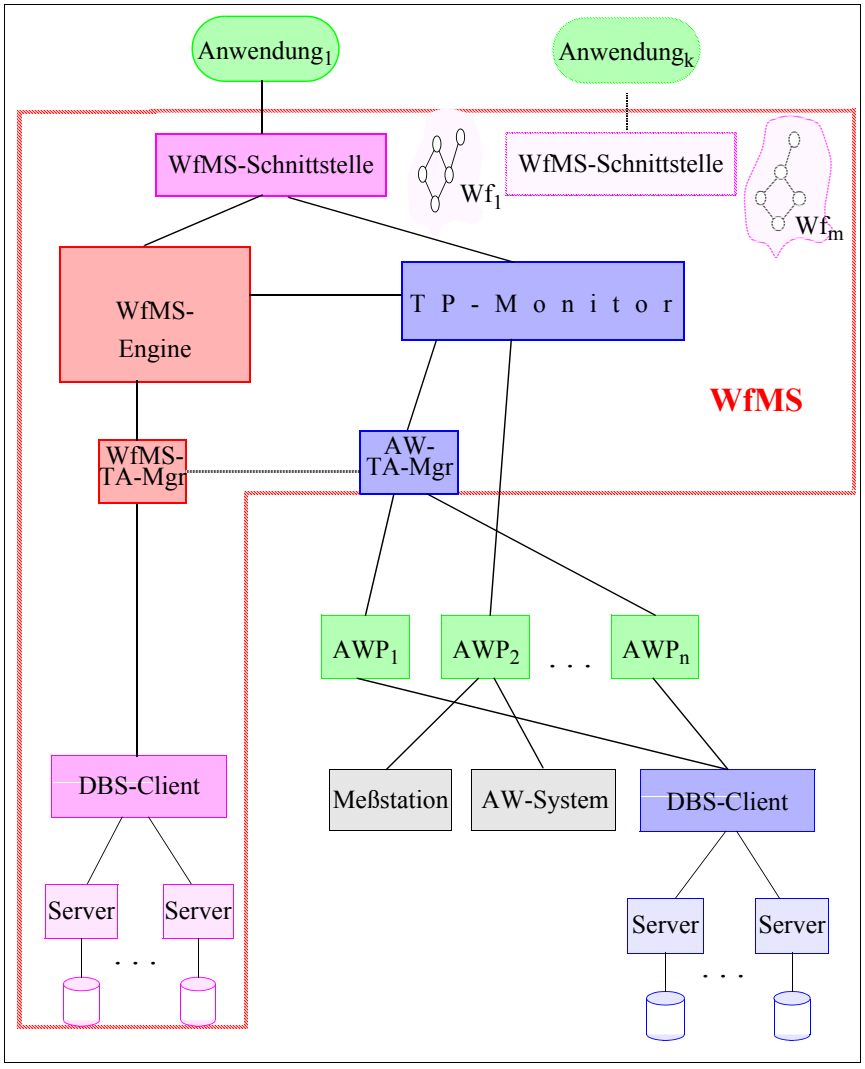
- **Zwei Arten von Datenhaltung erforderlich**
  - Kontrolldaten (durch WfMS (-DBMS) verwaltet)
  - Produktionsdaten (durch AW oder AW-DBMS verwaltet)
  - keine abgestimmten Kooperationskonzepte (Autonomie, Heterogenität)
  - **Achtung:** WfMS und AWs sind aus der Sicht ihrer DBMS Applikationen!



- **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

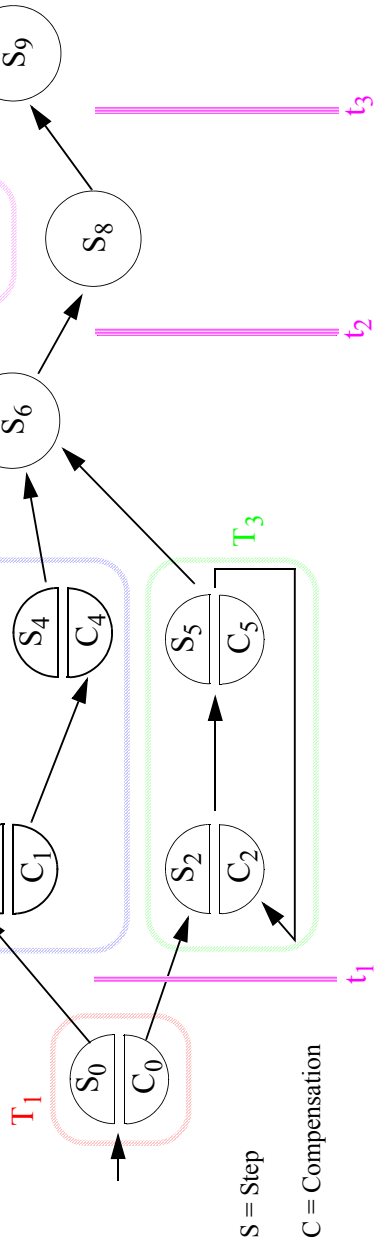
Funktionale Architektur von WfMS

- **Funktionale Architektur im Überblick**



Ausführung von Workflows (5)

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



- **Recovery-Aktionen bei**
  - Rücksetzen von T_2
  - Crash bei t_1, t_2, t_3
- **Was passiert mit den ungeschützten Aktionen?**

Ausführung von Workflows (6)

- **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

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

Stratifizierte Transaktionen

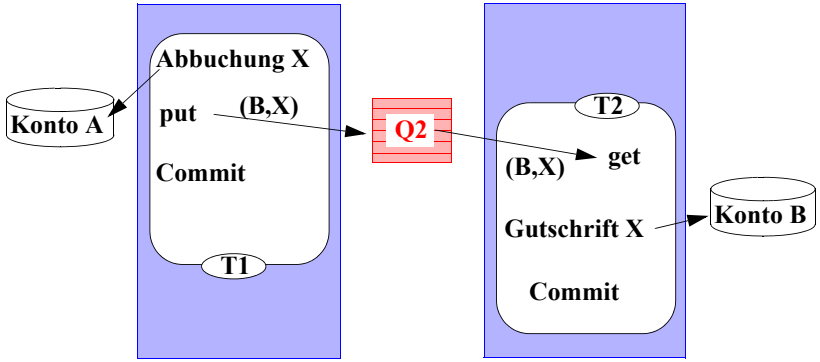
- **Phoenix-Verhalten von Workflows**
  - Recoverable Messaging erlaubt die Nutzung persistenter Warteschlangen (Queued Transaction Processing)

➥ Absicherung der Kommunikation zwischen WfMS-Komponenten

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

➥ 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)

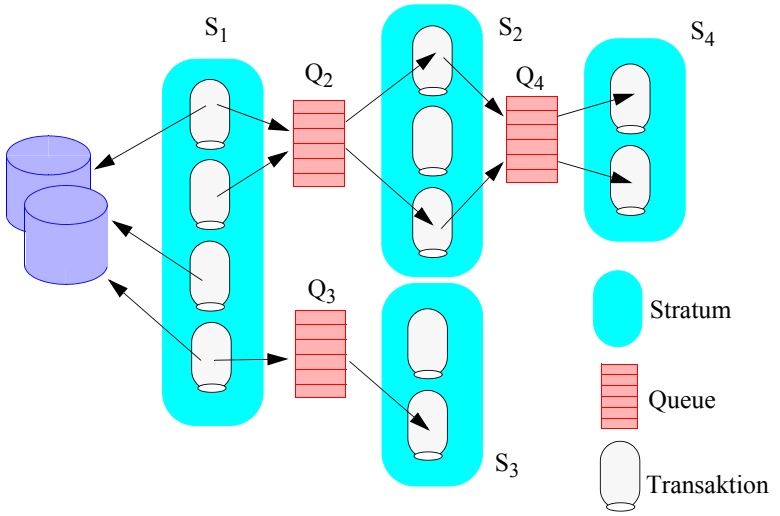
- **AW-orientierte Zerlegung der Transaktion T**
  - Erweiterung des Konzepts der Verkettung von TA durch Recoverable Messaging
  - Zerlegung in (möglicherweise verteilte) $T_1, \ldots T_n$;
  - Verkettung: jede T_i erhält persistente Warteschlange Q_i, aus der sie Anforderungen erhält, bestimmte Operationen 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 $S_1, \ldots, S_m$
  - ($S_i \subseteq T$ 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)