Kapitel 4

Transaktionale Workflow-Modelle

Inhalt

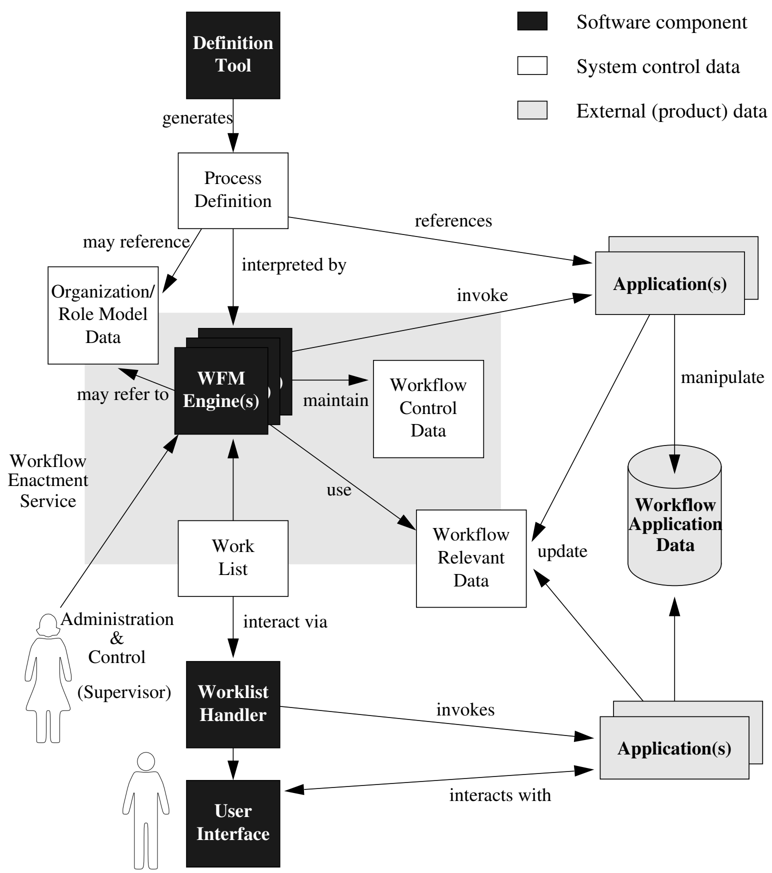
Workflow-Management (1)

- ❑ 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.*
- ❑ MOBILE[1]: Forschungsprototyp eines Workflow-Management-Systems
  - ➪ ***orthogonaler 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

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

Workflow-Management (3)

❑ **Workflow-Management-System** nach WfMC



ConTracts (1)

❑ ConTracts:[1] Erweiterung des Saga-Ansatzes um

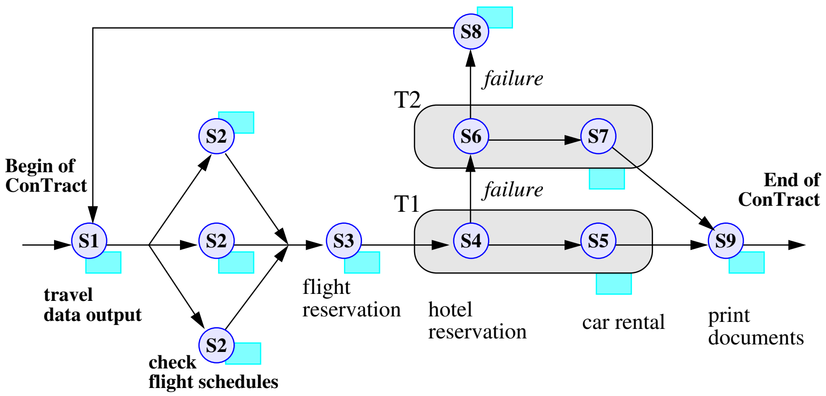
- reichere Kontrollstrukturen
  (Sequenz, Verzweigung, Parallelität, Schleife, etc.)
- getrennte Beschreibung von Sub-Transaktionen (**Steps**) und Ablaufkontrolle (**Skript**)
- Verwaltung eines persistenten **Kontextes** für globale Variablen, Zwischenergebnisse, Bildschirmausgaben, etc.
- Synchronisation zwischen Steps über Invarianten
- flexiblere Konflikt-/Fehlerbehandlung

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

ConTracts (2)

❑ Beispiel

➩ grafische 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: 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)

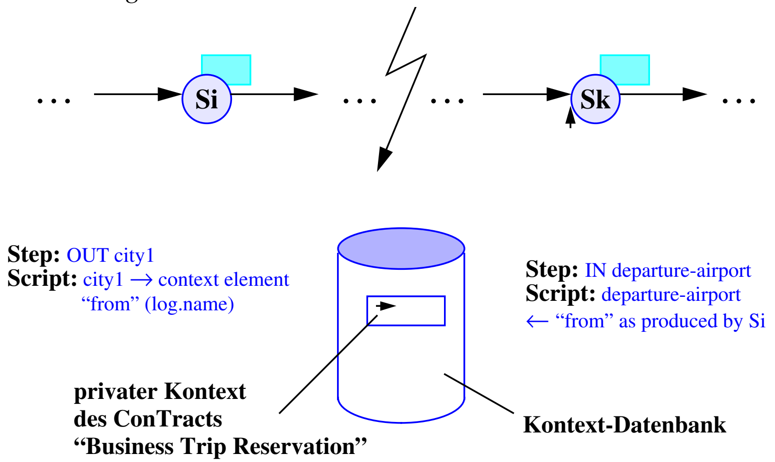
- Abhängigkeiten
  - Alternative zu obigem Beispiel

    T1 (S4, S5),
    DEPENDENCY(T1:abort[1] → begin:T1);
    /* erster Abort von T1 */
    DEPENDENCY(T2: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

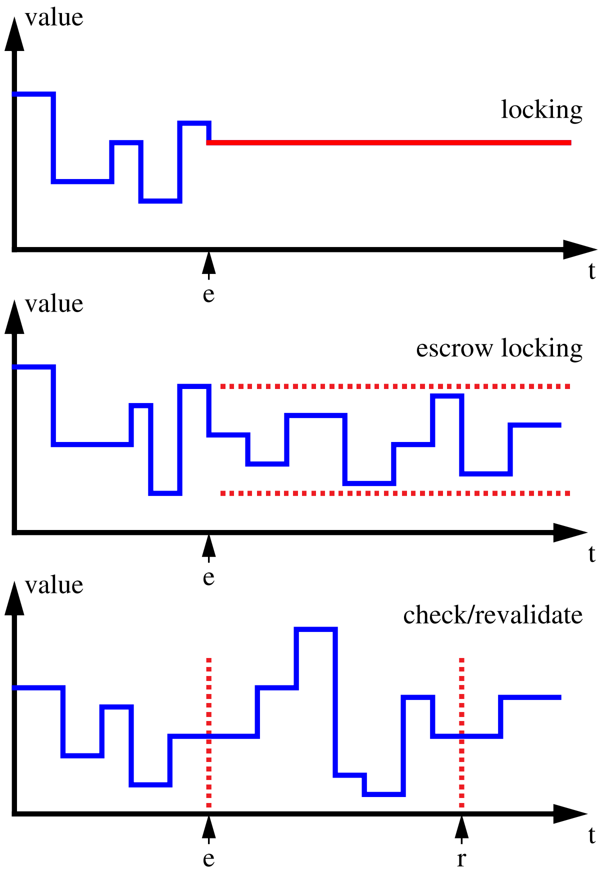
➩ Regeln;

- für jede(n) Step/Transaktion muß genau eine Kompensationstransaktion vorhanden sein
- mit dem Commit des Steps müssen alle Daten, die für die Kompensation gebraucht werden, berechnet und abgelegt sein
- lokale Daten, die für Komensationschritte 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 eine Kompensation nach k Wiederholungen immer noch nicht erfolgreich beendet werden konnte

ConTracts (9)

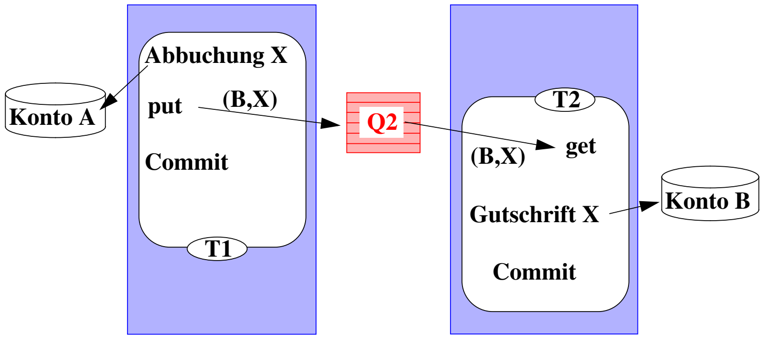
❑ Synchronisation mit *Invarianten*

➩ Möglichkeiten



Stratifizierte Transaktionen (1)

- Einschub: Recoverable Messaging



Zerlegung der globalen Transaktion "Buchung von Konto zu Konto"

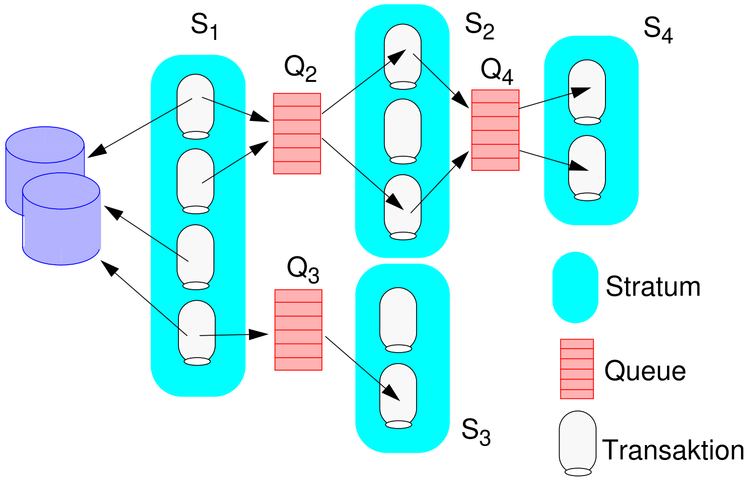
- ➪ Wichtigstes Prinzip: Enqueue / Dequeue wird jeweils innerhalb der Kontrollspähre der Schreib- / Lesetransaktion durchgeführt
- ➪ erfordert entsprechende Protokolle zwischen Queue-Manager und TA-Manager (mindestens 2PC)
- ➪ Grundlage asynchroner Transaktionsverarbeitung
- ➪ MOM: Message-oriented Middleware

Stratifizierte Transaktionen (2)

- AW-orientierte Zerlegung der Transaktion T
  - ➪ in 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 Resource-Manager (DBVS, MOM) in atomares Commit einbinden lassen (XA-Protokoll, 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 $\bigcup_{j=1}^{m} S_j = T$)
  - ➪ 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)

Stratifizierte Transaktionen (4)

- Vorteile:
  - im Vergleich zu T: früheres Commit der einzelnen Strata S_i; dies impliziert frühere Freigabe der Sperren und damit höhere Nebenläufigkeit
  - Antwortzeit für Benutzer: Ausführungszeit des Wurzelstratums
  - 2PC-Protokolle für alle Strata können weniger Nachrichten umfassen als das 2PC-Protokoll einer globalen Transaktion (*Kolokation* als mögliches Kriterium für Stratifikation von T: lokale 2PC-Nachrichten)

METEOR[1] (1)

❑ Schlüsselkonzepte:

➪ Task

➪ Koordination von Task-Ausführungen

➪ Korrektheit des Workflows

❑ Task

➪ Menge von extern sichtbaren Ausführungszuständen

➪ Menge von erlaubten Übergängen (Transitionen) zwischen den Zuständen

➪ Übergangsbedingungen

❑ Koordination

➪ Ausführung/Ausführbarkeit eines Tasks kann abhängen von

- Ausführungszuständen anderer Tasks, z. B.:
  “T1 darf nicht starten, bevor T2 beendet ist”,
  “Nach Commit von T1 muß T2 abgebrochen (Abort) werden”;

- Ausgabewerten anderer Tasks, z. B.:
  “T1 kann starten, wenn T2 einen Wert > 25 liefert”;

- externen Variablen
  im wesentlichen temporale Bedingungen, wie z. B.:
  “T1 kann nicht vor 9.00 Uhr MEZ gestartet werden”;

1. Rusinkiewics, M., Sheth, A.: Specification and Execution of Transactional Workflows, in: Kim, W. (Hrsg.): Modern Database Systems: The Object Model, Interoperability and Beyond, Addison-Wesley, 1994, S. 592-620.

METEOR (2)

❑ Korrektheit

➪ ***Fehleratomizität*** eines Workflows

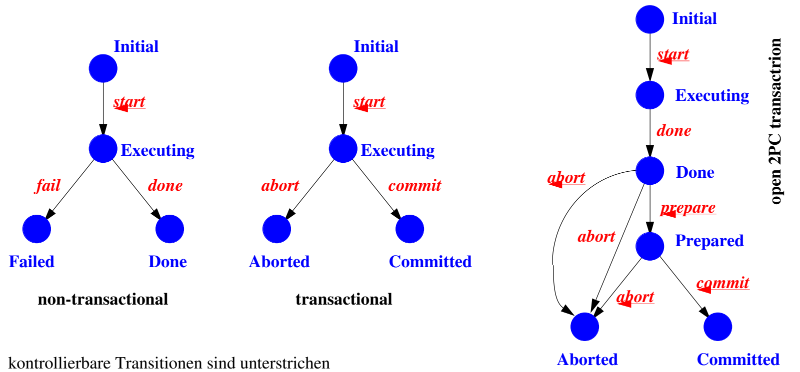
- Menge akzeptierbarer Terminierungszustände
- *committed acceptable termination states*:
  bei Erreichung wurde Workflow erfolgreich beendet
- *aborted acceptable termination states*:
  erlaubtes, aber nicht erfolgreiches Ende des Workflows;
  als Folge müssen alle bisher ausgeführten Tasks kompensiert werden

➪ ***Ausführungsatomizität*** eines Workflows

- ‘Serialisierbarkeit von Workflows’ ist zu strikt
- Synchronisation über Invarianten (Bedingungen)

METEOR (3)

- ❏ Task-Strukturen



kontrollierbare Transitionen sind unterstrichen

METEOR (4)

- ❏ Workflow-Spezifikation besteht aus:
  - Beschreibung der Strukturen aller beteiligter Tasks
  - Beschreibung der Eingaben/Ausgaben von Tasks und Filtern, sowie der Beziehungen zwischen Ein-/Ausgaben verschiedener Tasks
  - Vorbedingungen für jede kontrollierbare Transition eines jeden Tasks
- ❏ WFSL: Workflow Specification Language
  - Task-Klassen
  - Definition von *Compound Tasks*
  - Inter-Task Dependencies
  - *state dependencies* ...
    [L1, done] ENABLES [L2, start];
  - ... können mit *value dependencies* verbunden werden
    [L1, done] & (success(L1.output1)) & (outval4 > 5) ENABLES [L2, start];
  - Eingabe-/Ausgabe-Zuweisungen
    L1.output1 → L2.input1

METEOR (5)

❑ Beispiel für eine Workflow-Spezifikation

```
typedef char[2000] str;
constant int ERROR = 0;
constant int PARTIAL_SUCCESS = 1;

simple_task_type A_type SIMPLE_NON_TRANSACTIONAL
        (input str input1; output str output1);
simple_task_type B_type TRANSACTIONAL_OPEN2PC
        (input int input1; output int output1);
simple_task_type A_type TRANSACTIONAL_OPEN2PC
        (input int input1; output int output1);
task_class A_type A_class;
task_class B_type B_class;
task_class C_type C+class;
Filter int f1(str);
Filter int f2(str);

compound_task_tyope TRANS_BC COMPOUND_TRANSACTIONAL
        (input str input1);
{       B_class B; C_class C;
        int outB, outC;

1       [TRANS_BC, executing] ENABLES [B, start] %
                f1(TRANS_BC.input1) --> B.input1;
2       [TRANS_BC, executing] ENABLES [C, start] %
                f2(TRANS_BC.input1) --> C.input1;
3       [B, done] & [C, done] ENABLES [B, prepare] & [C, prepare] %
                B.output1 --> outB, C.output --> outC;
4       [B, prepared] & [C, prepared] & (outB > outC) ENABLES
                [B, commit] & [C, commit];
5       [B, committed] & [C, committed] ENABLES [TRANS_BC, commit];
6       [B, aborted] ENABLES [C, abort] & [TRANS_BC, abort];
7       [C, aborted] ENABLES [B, abort] & [TRANS_BC, abort];
}
task_class TRANS_BC BC_CLASS;

...
```


METEOR (6)

❑ Beispiel (Forts.)

```
...
compound_task_type WORKFLOW1 COMPOUND_NON_TRANSACTIONAL
        (input str input1; output str output1, int output2);
{       A_class A;
        BC_CLASS BC1;

8       [WORKFLOW1, executing] ENABLES [A, start] %
                WORKFLOW1.input1 --> A.input1;
9       [A, done] & (success(A.output1)) ENABLES [BC1, start] %
                A.output1 --> BC1.input1;
10      [BC1, committed] ENABLES [WORKFLOW1, done] %
                A.output1 --> WORKFLOW1.output1;
11      [A, failed] ENABLES [WORKFLOW1, fail] %
                ERROR --> WORKFLOW1.output2;
12      [BC1, aborted] ENABLES [WORKFLOW1, fail] %
                A.output1 --> WORKFLOW1.output1,
                PARTIAL_SUCCESS --> WORKFLOW1.output2;
}
```

METEOR (7)

- TSL: Task Specification Language
  - Makros zur Kommunikation der Zustände mit der Workflow-Engine
- Beispiel für eine Task-Spezifikation

```
Database_task (Sp_rec)
SPECIAL_REC Sp_rec;
{       EXEC SQL INCLUDE SQLCA;
        EXEC SQL BEGIN DECLARE SECTION;
                int infor;
        EXEC SQL END DECLARE SECTION;
        EXEC SQL WHENEVER SQLERROR goto Failed;
        TASK_EXECUTING();
        info = extract_info_from_rec(Sp_rec);
        EXEC SQL INSERT INTO INFO_table VALUES (:info);
        EXEC SQL COMMIT;
        TASK_COMMIT();
    Failed:
        EXEC SQL ROLLBACK;
        TASK_ABORT();
}
```

Makros

Zusammenfassung

- **Workflow-Management**
- **ConTracts**
  - Beispiel für ‘transaktionale’ Workflows
  - ausschließliche Betrachtung von ACID-Transaktionen als Teil-Aktivitäten
- **Kompensationsspähren**
  - Mengen semantisch zusammgehörender transaktionaler Teil-Aktivitäten
- **Strata**
  - *Recoverable Messaging* als Grundlage asynchroner Transaktionsverarbeitung
- **METEOR**
  - transaktionale Abhängigkeiten
  - Berücksichtigung nicht-transaktionaler Teil-Aktivitäten