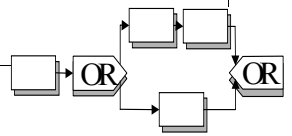
Das ConTract-Modell

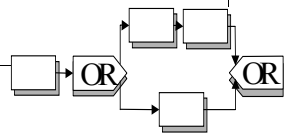
(Wächter & Reuter, Universität Stuttgart)

- Zielsetzung
- Programmier- und Fehlermodell
- Ausführung und Architektur
- Zusammenfassung



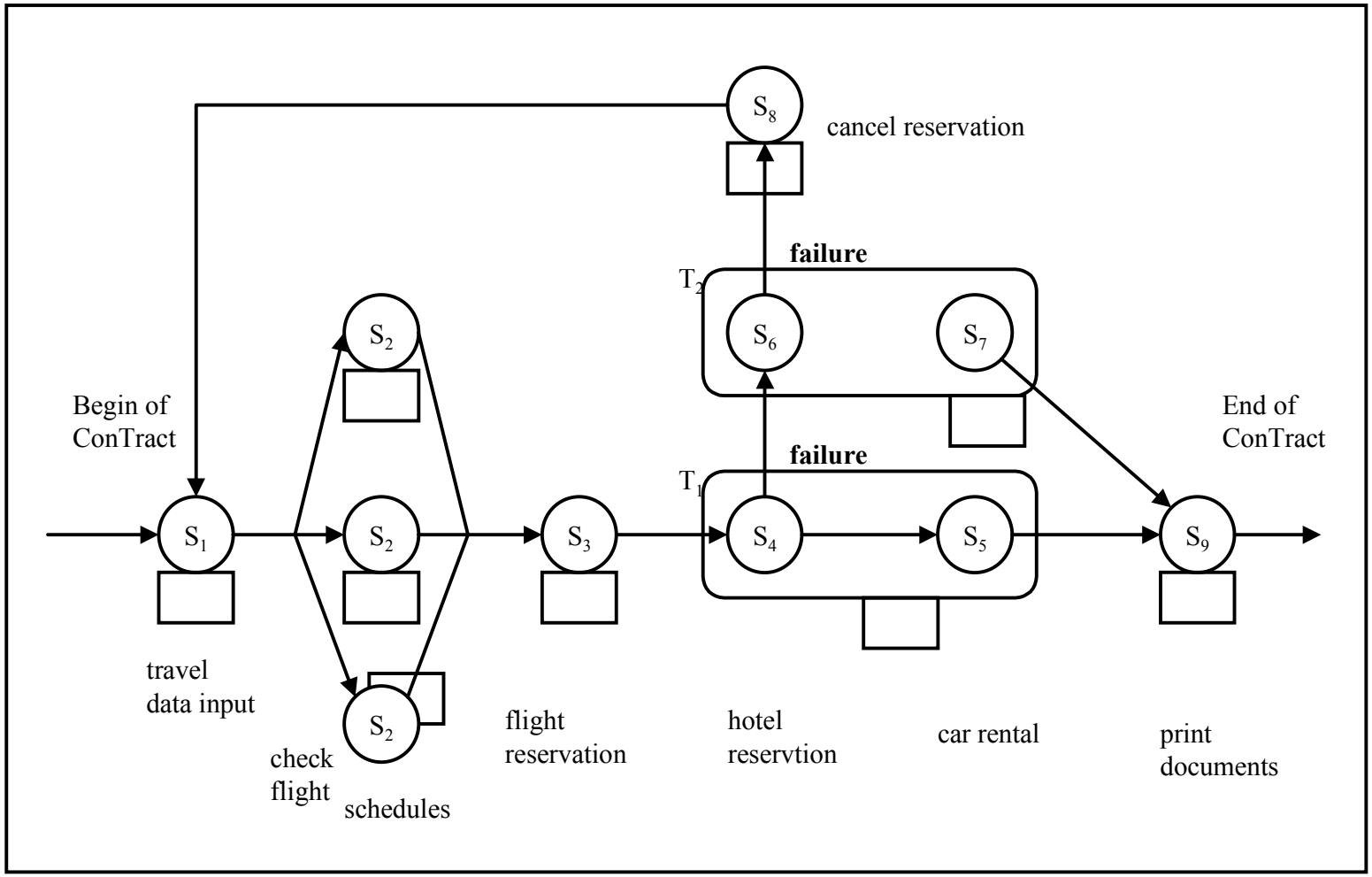
ConTract: Zielsetzung

- Bereitstellung eines Ausführungsmodells für die zuverlässige Abwicklung langlebiger und verteilter Workflows
- Forward Recovery eines Prozesses (“Phönix aus der Asche“)
- “Logisches“ Rollback durch Prozeß-Kompensation (falls Prozeß nicht fortsetzbar)
- Flexible Sperrverfahren (durch Isolationsprädikate)
- Prozeß-Migration (bei Rechnerausfall)



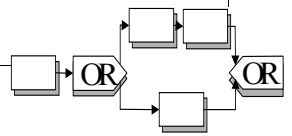
Das ConTract-Programmiermodell

- Ein *ConTract* (= Workflow/Prozeß) ist die konsistente und fehlertolerante Ausführung einer Anzahl von Aktionen (*Steps*), die in einer separat spezifizierten Kontrollflußbeschreibung (*Script*) sequentiell oder parallel miteinander verknüpft sind



Definition eines ConTracts: Step-Definition

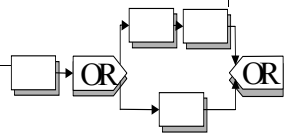
```
STEP Flight_Reservation

    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 ()
{   car*    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
       :
}
```



Definition eines ConTracts: Kontrollflußkonstrukte

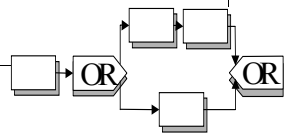
Sequenz bzw. Block	BEGIN <block> END <block>: Sequenz von Programmschritten oder anderen Kontrollflußanweisungen	**Fork und join paralleler Abschnitte**	PARBEGIN <block> PAREND
Verzweigung	IF <condition> <block> [ELSE <block>] ENDIF CASE <contextelement> <contextelement> : <block> [...[<contextelement> : <block>]] ENDCASE	**Parallele Schleife**	PARFOREACH (<contextelement> : <type> IN <set>) <block> ENDPARFOREACH

Schleife	WHILE <condition>
	<block>
	ENDWHILE
	
	FOR (<value-1> TO <value-2>)
	<block>
	ENDFOR
		Ablaufsteuerung	SUSPEND [<duration> | UNTIL <event>]
			STOP
			MIGRATE TO <node-id>



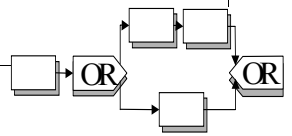
Kontrollfluß -Beispiel

```
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, seats, cost_limit);
   PARFOREACH( airline: EXECSQL select airline from ... ENDSQL )
       S2: Check_Flight_Schedule(in_context: airline, date, from, to, seats;
                                 out_context: flight_no, ticket_price );
   ENDPARFOREACH
  S3: Flight_Reservation(in_context: airline, flight_no, date, seats, ..);
  S4: Hotel_Reservation( in_context: „Cathedral Hill Hotel“;
                                     out_context: ok, hotel_reservation );
  IF ( ok[S4] ) THEN S5: Car_Rental( ... „Avis“ ... );
  ELSE BEGIN
       S6: Hotel_Reservation( ... „Holiday Inn“ ... );
       IF ( ok[S6] )
           THEN     S7: Car_Rental( ... „Hertz“ ... );
       ELSE  S8: Cancel_Flight_Reservation_&_Try_Another_One( .. ); ENDIF
         END
  ENDIF
  S9: Print_Documents( ... );
END_CONTROL-FLOW-SCRIPT
```



ConTract: Fehlermodell

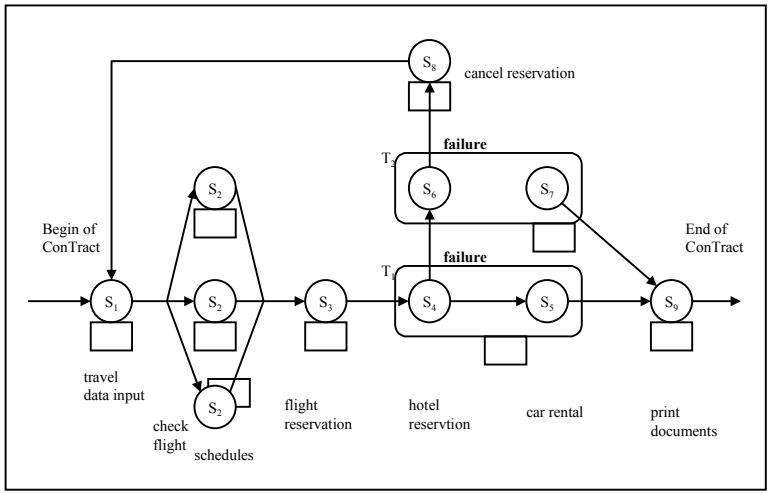
- Step ist (u.U. verteilte) ACID-Transaktion (Default)
- Mehrere Steps können zu ACID-Einheit zusammengefaßt werden
- Anwendungsspezifische Reaktionen auf Fehlersituationen
  - Wiederholung eines Steps (z.B. bei Synchronisationskonflikten)
  - Ausführung eines alternativen Steps
- Ein ConTract ist i. allg. *keine* ACID-Transaktion
  - Aufgabe der Atomarität
  - ConTract wird bei System-Fehler „eingefroren“ und später fortgeführt → Forward Recovery
  - Logisches Rollback eines ConTracts durch Kompensations-Steps (nur *explizit* durch Administrator oder Benutzer)
  - Keine strikte Isolation, sondern anwendungsspezifische Isolationsprädikate



ConTract: Transaktionsstrukturen (1)

ACID-Einheiten und Abhängigkeiten

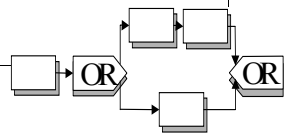
```
TRANSACTIONS
        T1 ( S4, S5 )
        T2 ( S6, S7 ) DEPENDENCY( T2 abort[1] | → begin T2 ) // first abort of T2
                      DEPENDENCY( T2 abort[2] | → begin S8 ) // second abort of T2
        ...
END_TRANSACTIONS
```



Weitere Notationsmöglichkeiten bzgl. Abhängigkeiten

(a = abort, b = begin, c = commit)

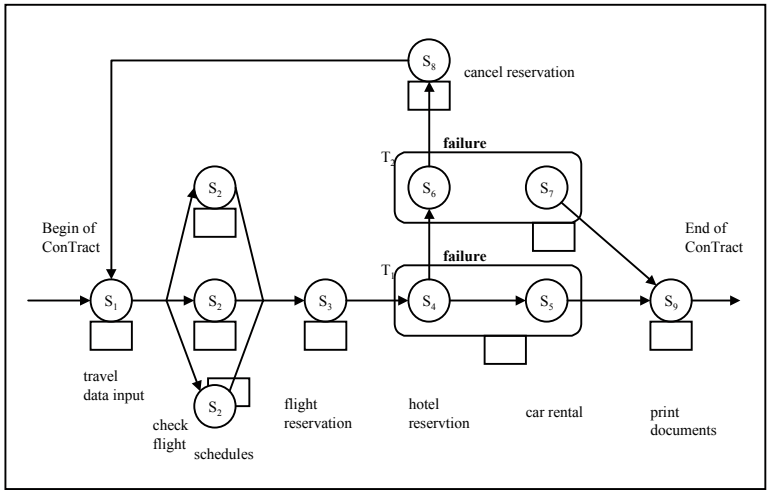
$T\ b \rightarrow b\ T_1 \ldots a \rightarrow b\ T_k\ a \rightarrow a\ T$ /* k alternatives for T */
$T_i\ c \rightarrow c\ T$
$T\ a[1] \rightarrow b\ T \ldots T\ a[n] \rightarrow b\ T_{rescue}$ /* retry T n times */



ConTract: Transaktionsstrukturen (2)

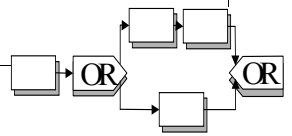
- Kompensationen
  - Start nur *explizit* durch Administrator oder Benutzer (z.B. bei endgültigem Scheitern eines Con-Tracts)

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



ConTract: Forward Recovery

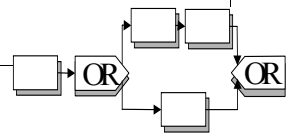
- Bei Systemfehlern
  - "Einfrieren" des Workflows
  - Fortführung nach Fehlerbehebung
- Kontext-Verwaltung
  - "Buchführung" über jeweiligen lokalen Verarbeitungszustand eines ConTracts
  - Kontextinhalte
    - Benutzereingaben, Bildschirmausgaben
    - Identifikatoren der globalen Objekte (z.B. in DB)
    - Eingabe- und Ausgabe-Variablen von Steps
  - Problematik?
  - Notwendige Voraussetzung auch für Migration
- Bei Forward Recovery Auslesen des ConTract-Zustands zum Fehler-Zeitpunkt und Wiederaufnahme
- Merke: Transaktionsfehler werden auf Step-Ebene (ACID-Transaktion) behandelt



Forward Recovery: Kontext-Verwaltung

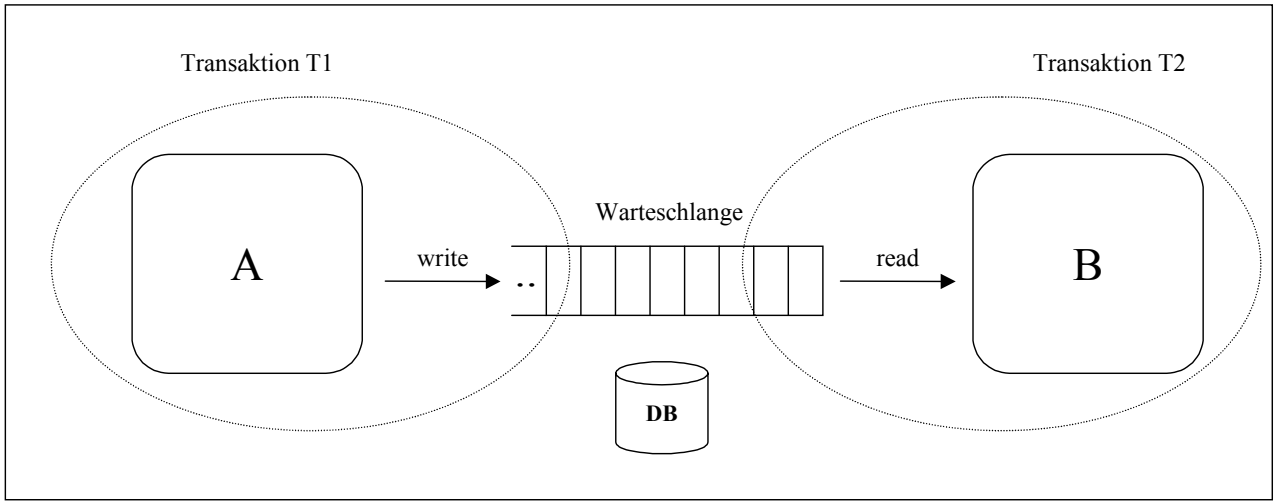
- Kein Kontext-Update, sondern Speicherung der gesamten ConTract-Chronologie
- Stabile Speicherung der Kontexte in transaktionsgeschützter SQL-Datenbank
- Fehlertolerante Übermittlung der Kontextvariablen über stabile Warteschlangen
- Beispiel für Kontext-Datenbank

Con-Tract Id	Erzeuger-Step			Kontext-Variable				Zeitstempel	
	Name	Version	Parfor-Index (Id des Paral-lelpfades)	Name	Typ	Version	Wert	Uhrzeit	Datum
4711	S1	1	0	start	string	1	Bonn	17:15	...
4711	S1	1	0	ziel	string	1	Rom	17:15	...
:	:	:	:	:	:	:	:	:	:
4711	S2	1	0	flug-nummer	string	1	LH560	17:20	...
	S2	1	1	flug-nummer	string	1	BA7788	16:00	
:	:	:	:	:	:	:	:	:	:



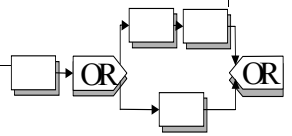
Stabile Warteschlangen

- Kopplung von Nachrichtenübermittlung an DBS
- Nachrichten werden persistent in DBS gehalten
- Zugriffe auf Nachrichten unter Transaktionsschutz
- Damit Status und Chronologie der Nachrichtenübermittlung zwischen Komponenten (z.B. Workflow-Engine und Anwendungen) rekonstruierbar und fehlertolerant



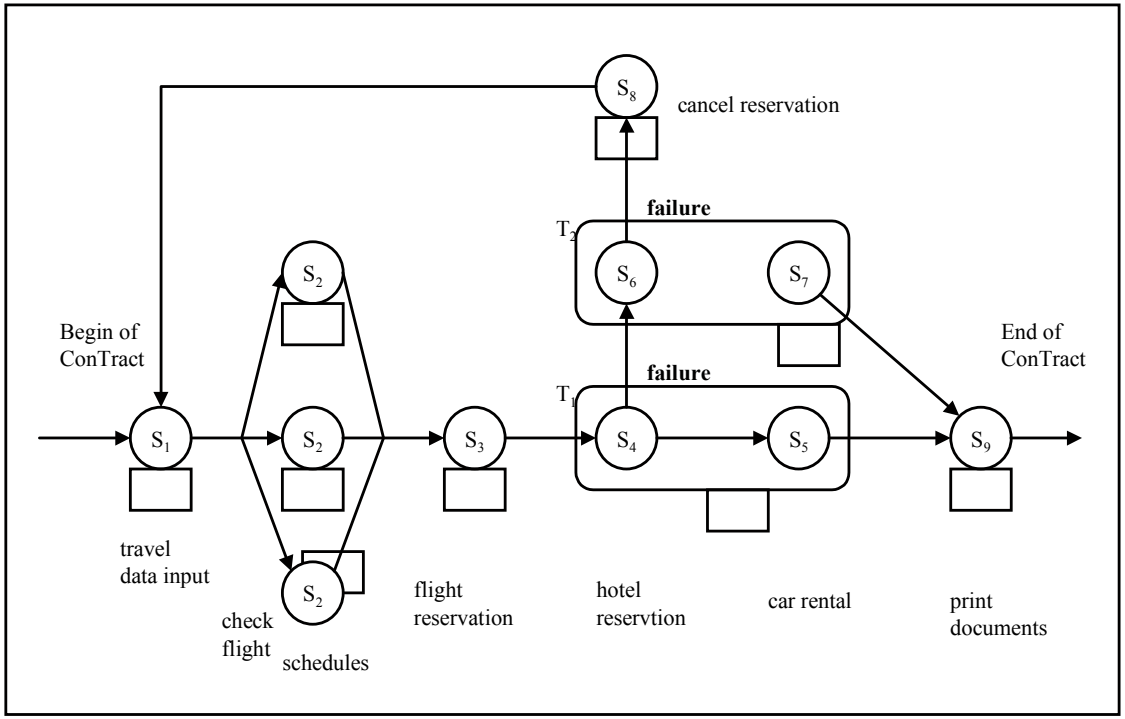
ConTract: Synchronisation mit Isolationsprädikaten

- Strikte Isolation bei langlebigen Prozessen zu restriktiv
- Beobachtung: Von einem Workflow benutzte Objekte müssen keineswegs gegen ALLE Arten von Änderungen (durch parallele Workflows) geschützt werden
- Lösungsvorschlag für ConTracts: *Isolationsprädikate*
  - Deklarative Spezifikation zulässiger Änderungsoperationen
  - Andere ConTracts dürfen Datenobjekte i.a. nur unter Nicht-Verletzung aktuell gültiger Prädikate ändern
- Prädikat-Typen für Steps
  - Eingangsprädikate P_{IE} bestimmen was Step tun darf
  - Ausgangs-Prädikate P_{IA} werden von Step dynamisch generiert und dienen Steps anderer Workflows als Eingangs-Prädikate



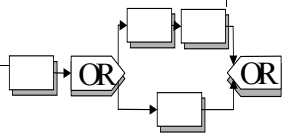
Isolationsprädikate: Beispiel

- Reisebuchung für Geschäftskunden
  - S1 überprüft, ob Abt.-Budget >= Reisekostenlimit
  - Falls ja, wird Ausgangs-Isolationsprädikat P_{IA} = (*budget >= limit)* generiert und als Eingangsprädikat alle weiteren Steps bekannt gemacht
  - Steps anderer ConTracts dürfen Wert von *budget* nur unter Nicht-Verletzung dieses Prädikats ändern (also Limit nicht unterschreiten)



Isolationsprädikate: Abstufungen

- **Locking**
  - Objekte des Prädikats werden exklusiv gesperrt
  - Für konkurrierende Anwendungen nur lesende Zugriffe erlaubt
  - entspricht “klassischem” Locking in DBMS
- **Mandatory** (häufigster Modus)
  - Nur Prädikat-konforme Operationen erlaubt
- **Check/Revalidate**
  - Alle Zugriffe weiterhin erlaubt
  - Prädikat wird erst bei neuem Zugriff des ConTracts auf Objekt überprüft
  - Sehr optimistische Strategie
  - Gefahr von Inkonsistenzen
  - Konfliktbehandlung erforderlich



Isolationsprädikate: Notierung

SYNCHRONIZATION_INVARIANTS_&_CONFLICT_RESOLUTIONS

S1: EXIT_INVARIANT(budget >= cost_limit),
POLICY: mandatory;

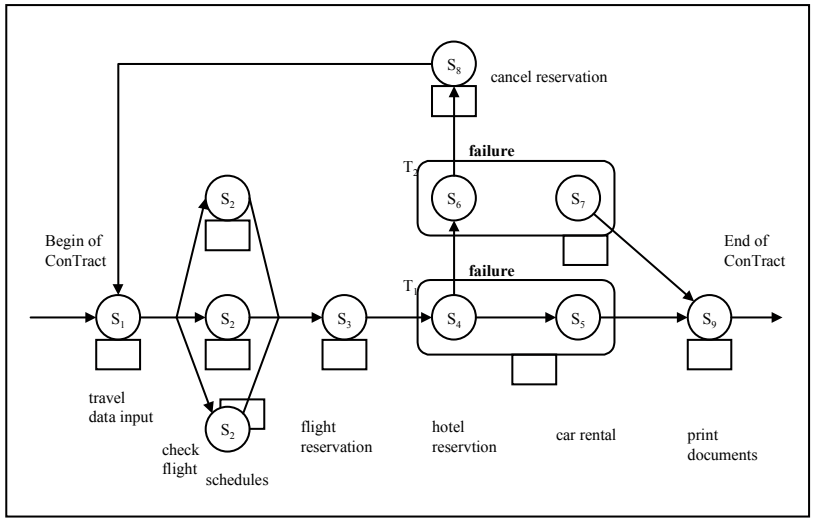
S3: ENTRY_INVARIANT(cost_limit > ticket_price); // Ticket darf Limit nicht erreichen
CONFLICT_RESOLUTION: S8: Cancel_Reservation(..);// andernfalls Abbruch der Reservierung

// ... “budget” wird von S3 um “ticket_price” herabgesetzt ...

EXIT_INVARIANT(budget > cost_limit - ticket_price); // ticket_price vom limit abziehen
POLICY:mandatory;

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

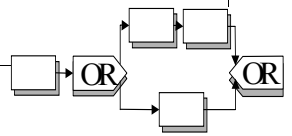
S5, S7: ENTRY_INVARIANT(car_price < budget),
CONFLICT_RESOLUTION: S10:
Call_Manager_to_Increase_Budget (..); :



Isolationsprädikate: Umsetzung in Trigger

- Übersetzung eines Prädikats P_{IE}/P_{IA} in DB-Trigger
- Rollback des Steps bei Verletzung des Prädikats
- Trigger zur Realisierung des Prädikats *abteilung.budget >= limit*

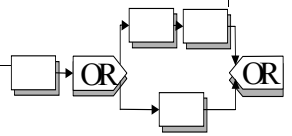
```
CREATE RULE   check_isolation_condition_on_budget
AFTER         UPDATE (budget) OF abteilung
WHERE         abteilung.name = "ABC"     /* Wert der Variablen :abtname*/
AND           abteilung.budget < 4 000   /* Wert der Variablen :limit  */
                                         /* jeweils am Ende von S1.    */
EXECUTE       PROCEDURE reject_operation
```



Isolationsprädikate: Konfliktsituationen

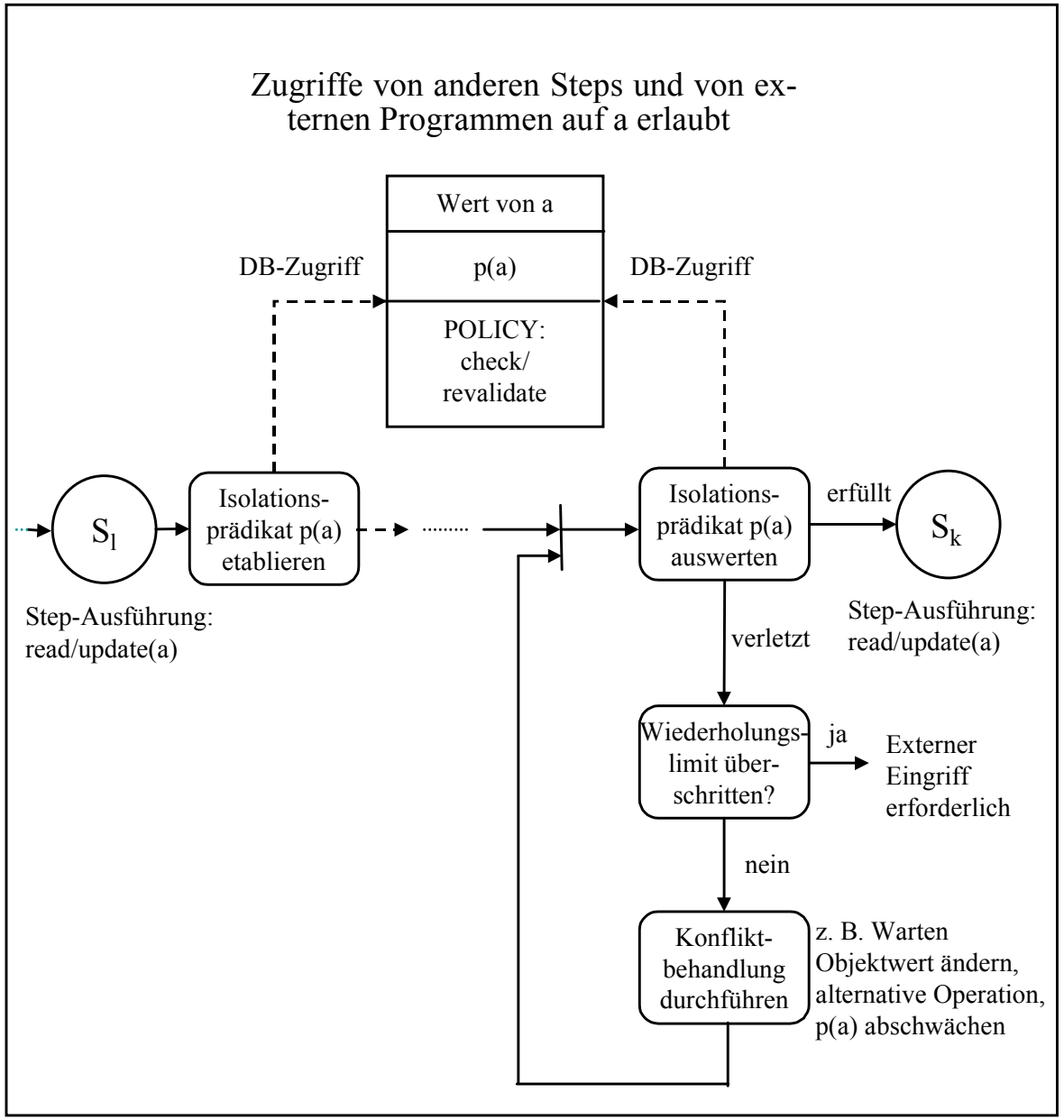
Konflikt / Reaktionsmöglichkeit	P_{IE}-Auswertung nicht möglich, da Objekt(e) gesperrt	Isolationsbedingungen P_{IE} bzw. P_{IA} nicht erfüllt
Im Skript vordefinierte Konfliktbehandlung aktiv.	+	+
Warten - mit Vormerkung - mit Zeitlimit - mit Suspendierung des ConTract	 + + o	 o o o
Abbruch und/oder Wiederholung des Step	+	+
Abbruch des ConTract (Kompensation)	-	o
Manueller Eingriff des Systemverwalters	-	o
Änderung der beteiligten Objekte bzw. Prädikate	o	+

Legende: + = gut geeignet; o = bedingt geeignet; - = nicht sinnvoll



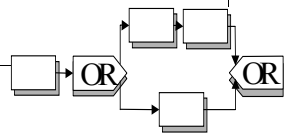
Konfliktbehandlung bei Check/ Revalidate-Modus

- Alle Zugriffe weiterhin erlaubt; Prädikat wird erst bei neuem Zugriff des ConTracts auf Objekt überprüft

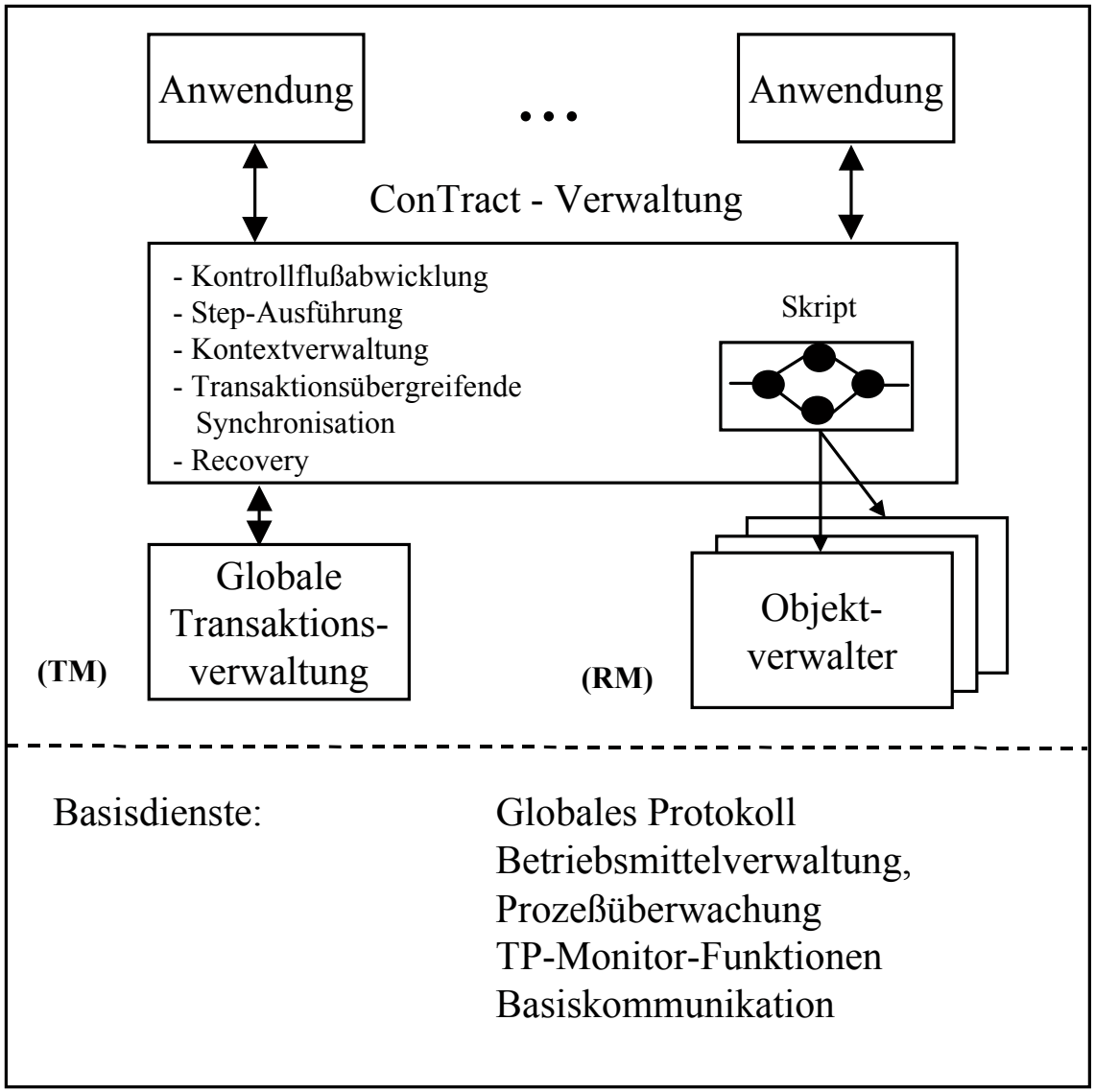


ConTract: Architektur und Ausführungsmodell

- Distributed Computing Environment (DCE) als Middleware
- X/Open DTP für Realisierung verteilter ACID-Transaktionen (2PC-Protokoll)
- Erweiterung der X/Open DTP Architektur durch zusätzliche Schicht (*ConTract-Verwaltungsschicht*) zwischen Anwendungen, Ressourcen-Managern (RM) und Transaktions-Manager (TM)
- Apricots: Prototypische Implementierung an Universität Stuttgart

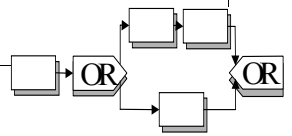


ConTract: Komponenten und Einbettung in DTP-Modell

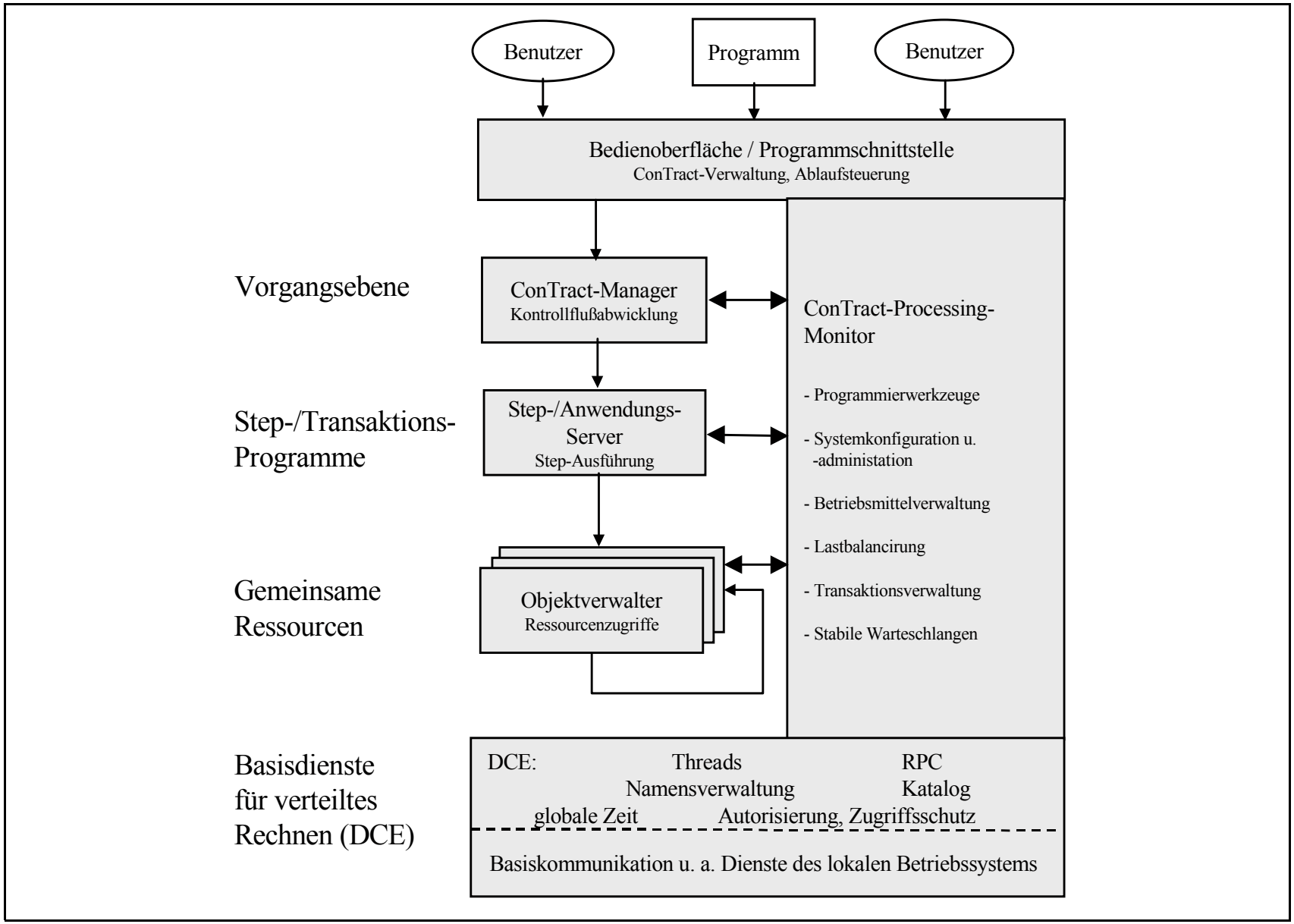


ConTract-Verwaltung: Komponenten

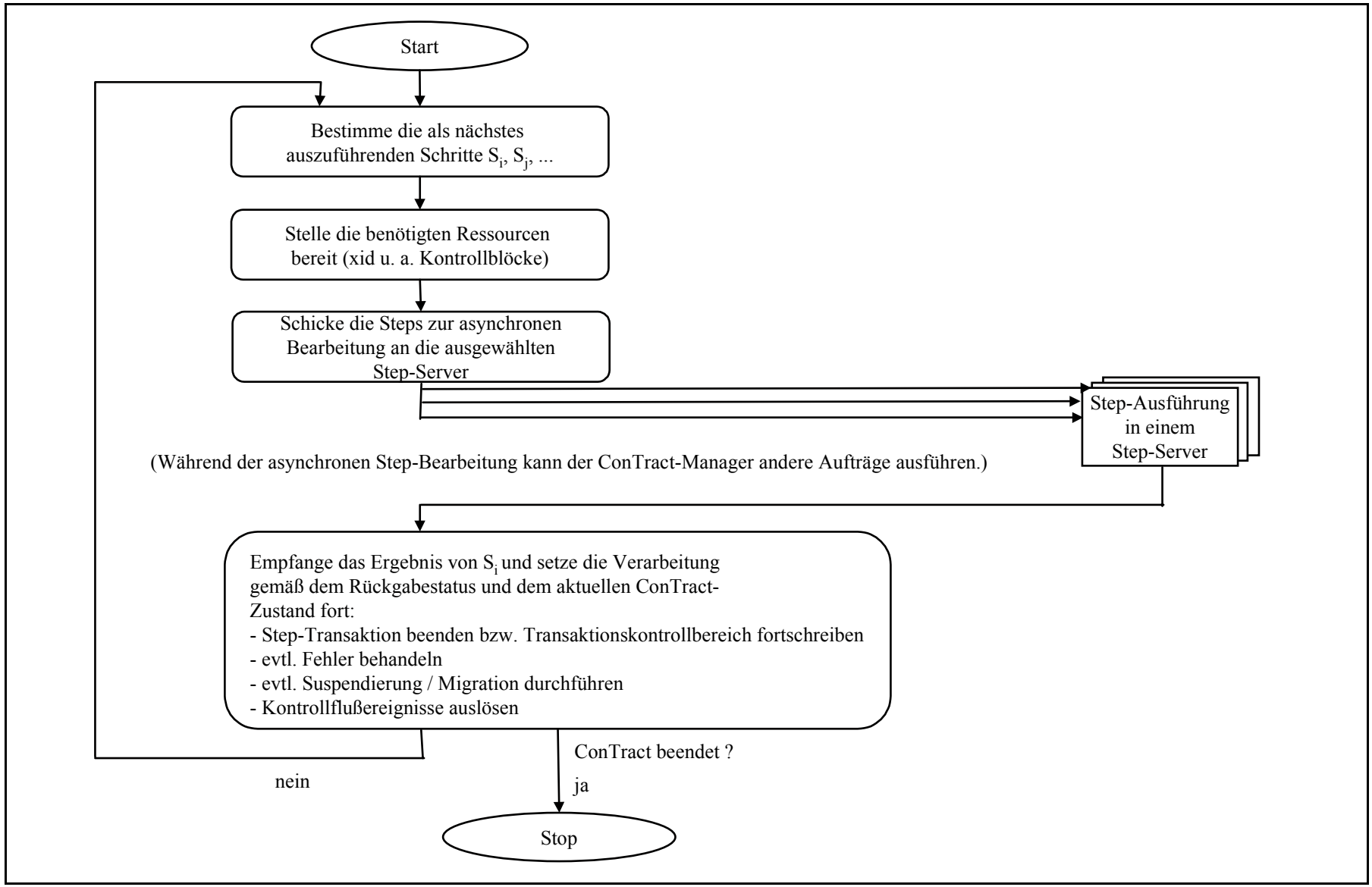
- ConTract-Manager (Skript-Laufzeitsystem)
  - Kontrollflußabwicklung eines ConTracts
  - Aktivierung und Überwachung der Steps
  - Kommunikation mit TM für Step-übergreifende Transaktionen
  - Kontext-Verwaltung
  - Forward Recovery eines ConTracts
- Step-Server
  - Ausführung der Steps
  - Auswertung der Isolationsprädikate
  - Kommunikation mit TM für transaktionsgeschützte Step-Ausführung
- ConTract-Processing Monitor (pro Knoten)
  - Ablaufübergreifende Dienste (Lastbalancierung innerhalb der Step-Server-Klassen)
  - Stabile Warteschlangen für Aufträge und Ergebnisse bei Kommunikation ConTract-Managern und Step-Servern



ConTract-Verwaltung: Schichtenarchitektur

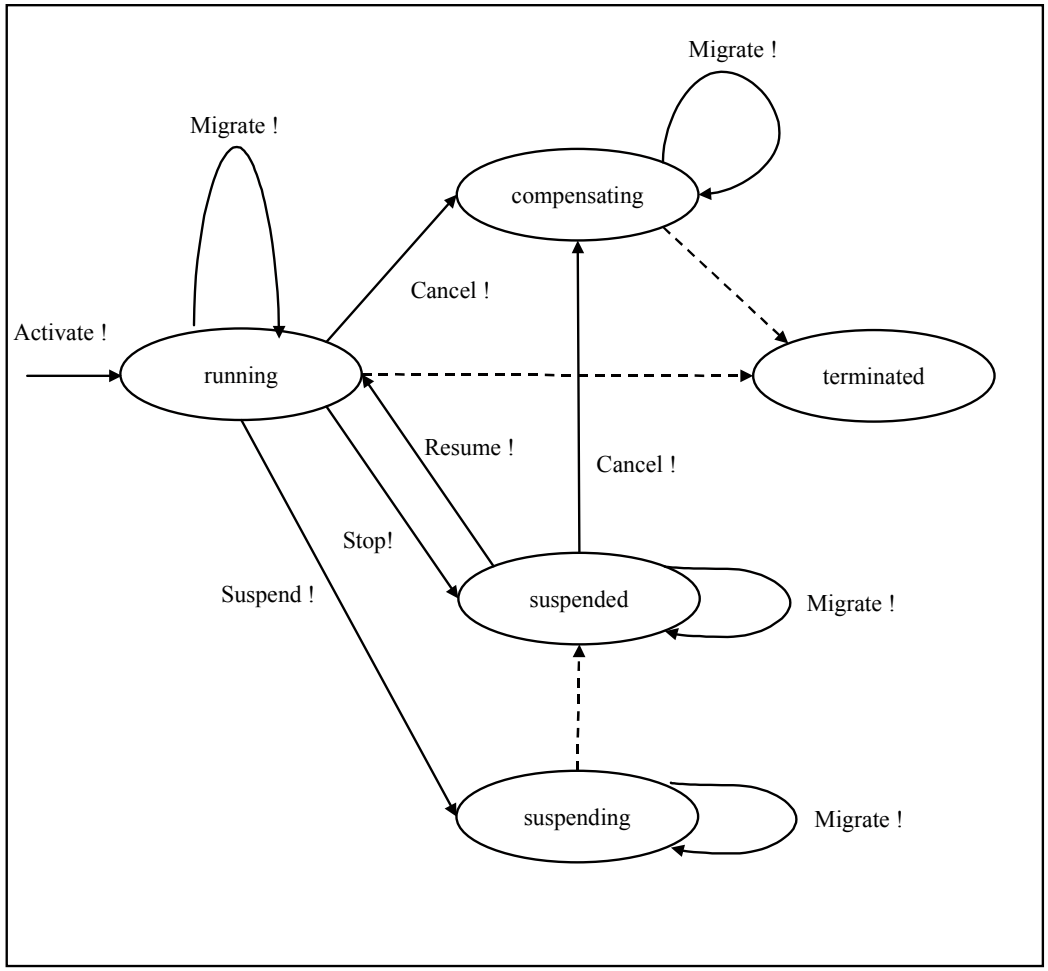


Skript-Abarbeitung durch ConTract-Manager



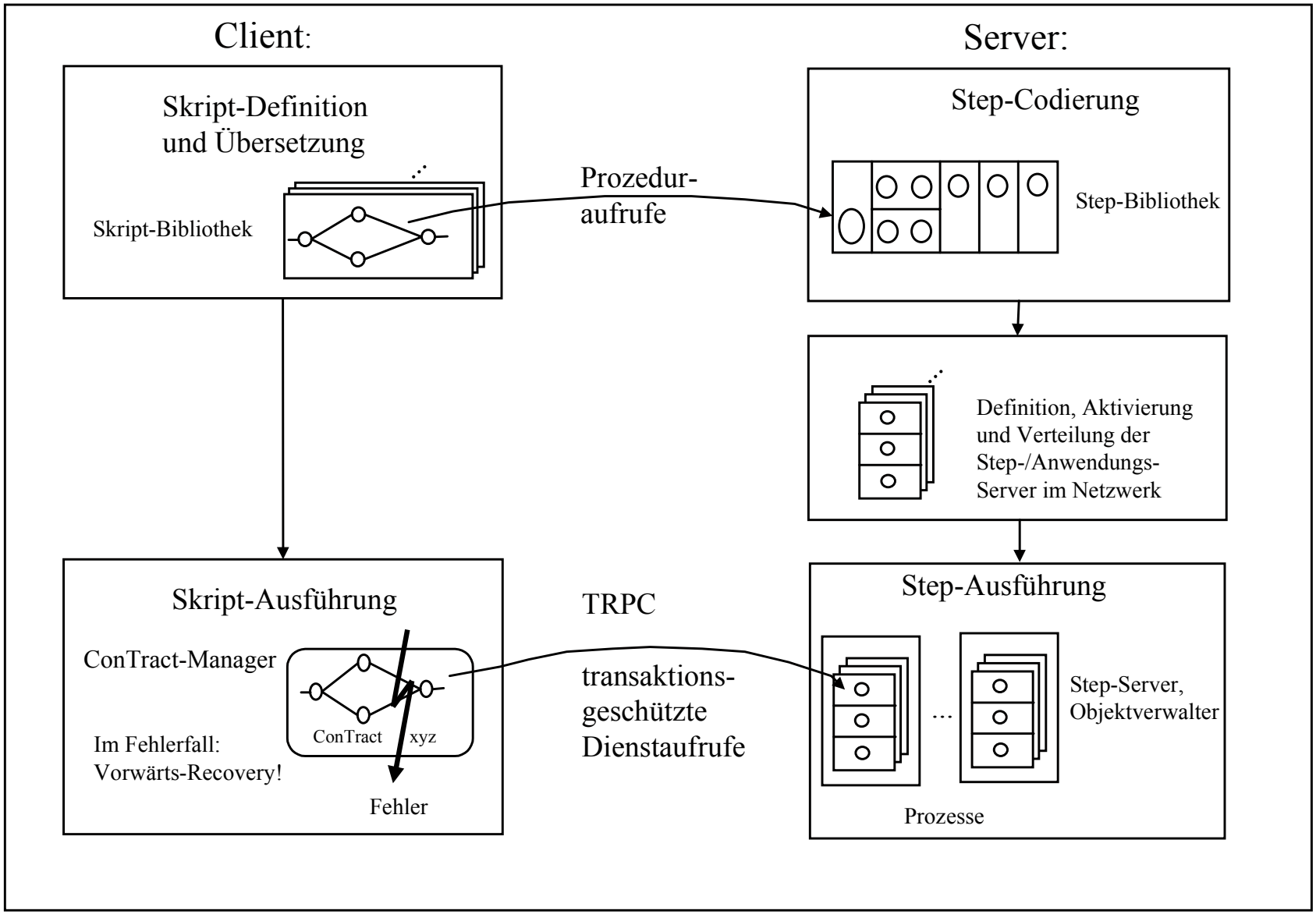
ConTract: Globale Ereignisse

- **suspend** (cid)
  - Anhalten des ConTracts für unbestimmte Zeit
  - System wartet vor Suspendierung Ende aller momentan aktiven Steps ab
  - Bis dahin eingetretene Aktivierungsereignisse für nachfolgende Steps erst nach Wiederaufnahme wirksam
- **stop** (cid)
  - Sofortiger Stop des ConTracts durch radikalen Abbruch aller offenen Steps
- **resume** (cid)
  - Setzt angehaltene Ausführung fort
- **migrate** (cid to-host)
  - Verlagert Ablaufkontrolle auf anderen Rechner im Netzwerk



- **cancel** (cid)
  - Bricht ConTract ab und leitet Kompensation aller bisher durchgeführten Programmschritte ein

Zusammenspiel Step-Server / ConTract-Manager



ConTract: Zusammenfassung

- System zur zuverlässigen Ausführung langlebiger und verteilter Workflows
- Forward Recovery von Workflows bei Systemfehlern
- Rollback mit Kompensation nur auf expliziten Benutzerwunsch
- Abschwächung der strikten Isolation durch Einführung von Isolationsprädikaten
- ConTract-Architektur auf der Basis von DCE und X/OPEN DTP
- Einschränkungen
  - Kontext-Verwaltung abhängig von API zu Anwendungsprogrammen
  - Keine Behandlung *logischer* Fehler