Kapitel 3

Kooperationsmodelle

Inhalt

- Serialisierbarkeit vs. Kooperation
- Kooperative Transaktionshierarchien
  - Benutzerdefinierte Korrektheit
  - Partielle Recovery
- CONCORD
  - Versionsverwaltung
  - Transaktionsverwaltung
  - Designflow-Management
  - Kooperationskontrolle
- Zusammenfassung

Serialisierbarkeit vs. Kooperation

❑ Kooperatives Problemlösen

➪ Charakteristika

- Anwendungsbereiche hoher Komplexität
- Auftragszerlegung / Teamorientierung / schrittweise Verfeinerung
- langlebige, interaktive Aktivitäten / werkzeugbasierte Verarbeitung
- komplexe Objektstrukturen / Versionen

➪ Probleme

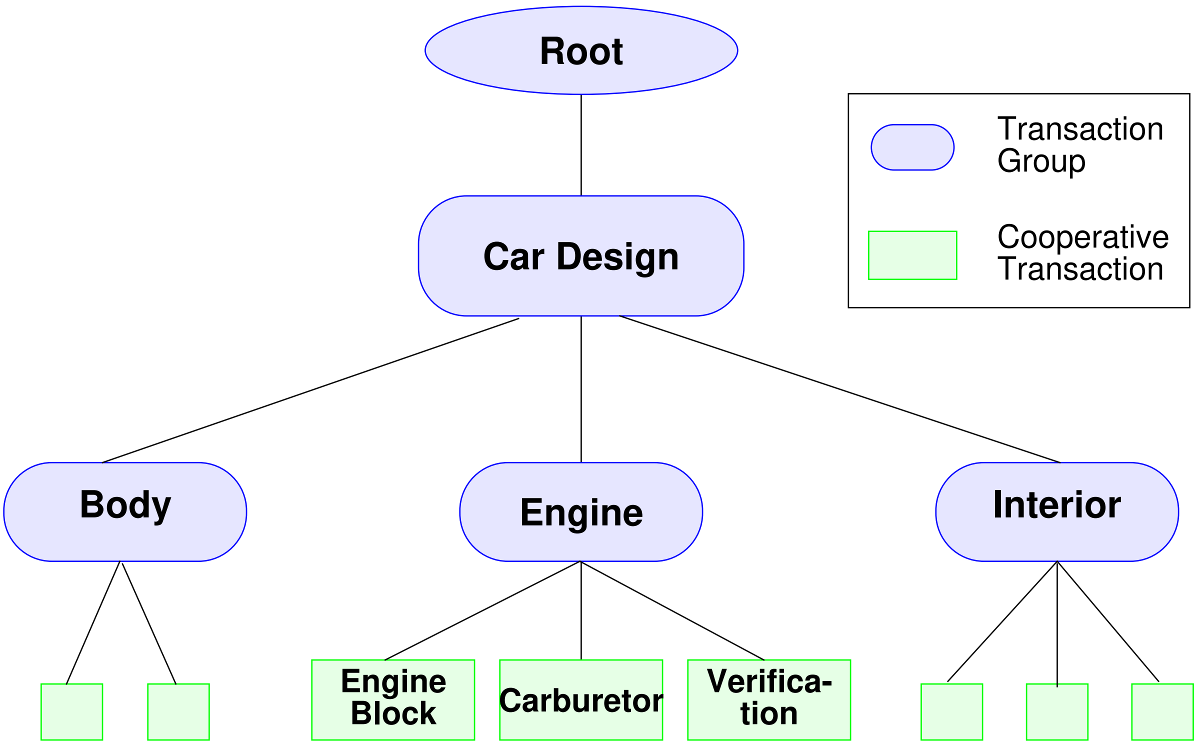
- Isolation vs. Kooperation
- vollständige vs. abgestufte Recovery

➪ Ansätze

- Relaxierung der Serialisierbarkeit
  - einfachstes Beispiel: Isolationsebenen
- Benutzerdefinierte Korrektheit
  - Beispiel: Kooperative Transaktionshierachien
- Kooperationskontrolle ‘oberhalb’ von ACID-Transaktionen
  - Beispiel: CONCORD

Kooperative Transaktionshierarchien[1] (1)

- Charakteristika
  - Hierarchische Organisation der Ablaufeinheiten
    - Kooperative Transaktion (CT):
      - führt Operationen auf dem zugehörigen Datenbestand aus
      - Blatt der Hierarchie
    - Transaktionsgruppe (TG):
      - Kollektion von CTs oder TGs
      - repräsentiert Task
      - innerer Knoten der Hierarchie
    - Beispiel



1. Nodine, M.H., Ramaswamy S., Zdonik, S.B.: *A cooperative Transaction Model for Design Databases*, in: Elmargarmid, A.K.: Database Transaction Models for Advanced Applications, Morgan Kaufmann, San Mateo, 1992

Kooperative Transaktionshierarchien (2)

❑ Charakteristika (Forts.)

➪ Korrektheit

- *benutzerdefinierte Korrektheitskriterien* auf TG-Ebene
- Nutzung von *Grammatiken* zur Beschreibung korrekter Historien
- 2 Möglichkeiten der Überprüfung der Korrektheit
  - für jeden Teilnehmer (direkter Nachfolger) einer TG wird seine vollständige Sequenz von Operationen geprüft
  - *Submit* einzelner Operationen
- *Submit*-Ansatz erfordert die Erkennung gültiger Präfixe von korrekten Historien

➪ Objektkopien

- jede TG hält private Kopien von den Objekten, die durch ihre Teilnehmer manipuliert werden
  - eine TG nimmt Kopie eines Objektes in privaten Objekt-Pool auf, sobald ein Teilnehmer das erste mal auf dieses Objekt zugreift
  - die Objekte im Objekt-Pool werden durch die von den Teilnehmern initiierten Operationen manipuliert (Beachte: Korrektheit)
  - beim Commit einer TG werden ihre Objekte von ihrer Vater-TG ‘übernommen’
  - durch das Commit der Wurzel-TG werden die Ergebnisse der Bearbeitung in die eigentliche (globale) Datenbank eingebracht

Kooperative Transaktionshierarchien (3)

❑ Charakteristika (Forts.)

➪ Operationsbezogene Recovery

- einzelne Operation kann auf explizite Anforderung ‘zurückgenommen’ werden (Abort von Operationen)
  - dies bedeutet, dass ihre Effekte sowie die Effekte von ihr abhängiger, weiterer Operationen gelöscht werden
  - kann mehrere Ablaufeinheiten betreffen
  - kein vollständiges Zurücksetzen von Ablaufeinheiten
  - erfordert die Verwaltung von Abhängigkeiten zwischen den Operationen
  - aufgrund von Abhängigkeiten zurückgesetzte Operationen können (teilweise) nach der Recovery und aufgrund expliziter Anforderung wiederholt werden

➪ Checkpoints

- ein Teilnehmer kann den Nettoeffekt seiner Operationen, für die noch kein Abort oder Checkpoint durchgeführt worden ist, zum Vater seiner TG propagieren

➪ Operationen

- **<m, o, O>**
  - m: Teilnehmername
  - o: Operationsbezeichner
  - O: Objekt-ID
- Submit-Ansatz!

➪ Historie (einer TG)

- partiell geordnete Folge von Operationen, die auf den Versionen im Objekt-Pool der TG ausgeführt wurden

Kooperative Transaktionshierarchien (4)

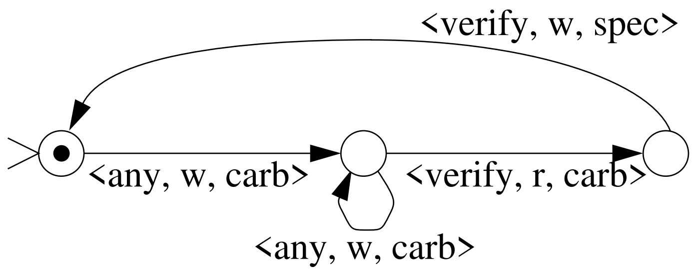
- Korrektheit
  - LR(0)-Grammatiken
    - "left-to-right scan of the input using 0 symbols of lookahead"
    - kontext-frei
  - Patterns
    - beschreiben Folgen von Operationen, die auftreten müssen
  - Conflicts
    - beschreiben Folgen von Operation, die nicht auftreten dürfen
  - Definition
    - Eine Historie is korrekt, wenn jedes Pattern, das während der Historie aktiv war, das Pattern-Kriterium und jeder Conflict, der während der Historie aktiv war, das Conflict-Kriterium erfüllen
    - Pattern-Kriterium
      - sei H_i der Teil der Historie, der generiert wurde, während die Pattern-Grammatik P_i aktiv war; sei S_i die Projektion der Operationen in H_i, für die P_i relevant ist; P_i erfüllt das Pattern-Kriterium, wenn S_i ein gültiges Wort der Grammatik P_i ist;
    - Conflict-Kriterium
      - betrachte Conflict-Grammatik C_j und H_j, S_j analog; C_j erfüllt das Conflict-Kriterium, wenn S_j kein gültiges Wort der Grammatik C_j ist;

Kooperative Transaktionshierarchien (5)

- Korrektheit (Forts.)
  - Beispiel

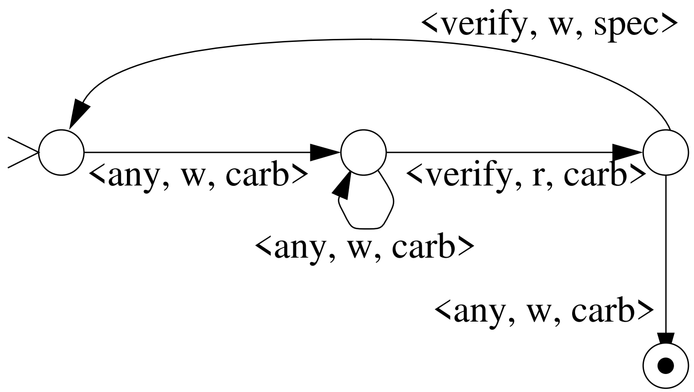
P_1:

S ← <any, w, carb> **A** | ε
A ← <any, w, carb> **A** | <verify, r, carb> **B**
B ← <verify, w, spec> **S**



C_1:

S ← <any, w, carb> **A**
A ← <any, w, carb> **A** | <verify, r, carb> **B**
B ← <any, w, carb> | <verify, w, spec> **S**



Kooperative Transaktionshierarchien (6)

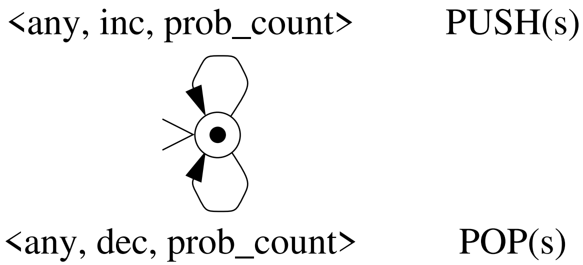
- Korrektheit (Forts.)
  - Beispiel

P_2:

S ← **B**

B ← **AB** | **A**

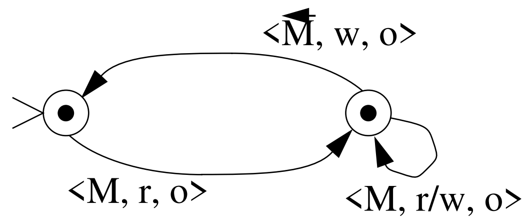
A ← <any, inc, prob_count> **B** <any, dec, prob_count> |
<any, inc, prob_count> <any, dec, prob_count>



P_S:

S ← <M, r, o> **A** | ε

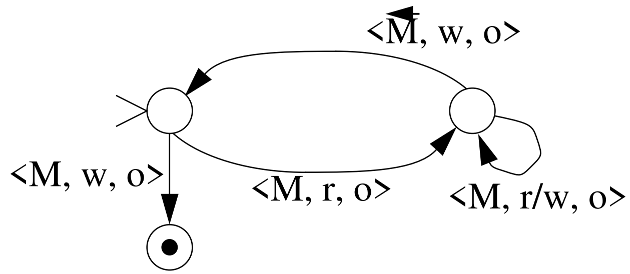
A ← <$\overline{\text{M}}$, w, o> **S** | <M, r/w, o> **A** | ε



C_S:

S ← <M, w, o> | <M, r, o> **A**

A ← <$\overline{\text{M}}$, w, o> **S** | <M, r/w, o> **A**



Kooperative Transaktionshierachien (7)

❑ Korrektheit (Forts.)

➩ Algorithmus

- Pattern-DPDA[1]
  - gibt es für die eingereichte (*submit*) Operation keine ausgehende Kante vom aktuellen Zustand, so ist dieser DPDA nicht relevant für die Entscheidung (***ignore***; BEACHTE: andere DPDAs können durchaus trotzdem relevant sein!)
  - bedeutet die eingereichte Operation einen Übergang zu einem Dead-End, so ist diese abzulehnen (***reject***)
  - bedeutet die eingereichte Operation einen Übergang zu einem Zustand, von dem aus ein erlaubter Endzustand erreicht werden kann, so ist die Operation zu akzeptieren (***accept***)
- Conflict-DPDA
  - keine ausgehende Kante → ***ignore***
  - Übergang in Endzustand → ***reject***
  - Übergang in Zustand, von dem aus Nicht-Endzustand erreicht werden kann → ***accept***
- Vorgehensweise
  - eine eingereichte Operation wird akzeptiert, wenn keine der relevanten Grammatiken sie ablehnt
  - eine eingereichte Operation wird abgelehnt, wenn eine der relevanten Grammatiken sie ablehnt
  - wenn eine Operation akzeptiert wird, wird für jede Grammatik der aktuelle Zustand (Präfix) entsprechend fortgeschrieben

1. Theorie: Für jede LR(0)-Grammatik gibt es äquivalenten DPDA (Deterministic Push Down Automaton) und umgekehrt!

Kooperative Transaktionshierachien (8)

- Beispiel
  - Grammatiken

P_3	**S** ← <Carburetor, r, carb> **A** | ε **A** ← <Carburetor, w, carb> **S** | <Carburetor, r/w, carb> **A** | ε
C_3	**S** ← <Carburetor, w, carb> | <Carburetor, r, carb> **A** **A** ← <Carburetor, w, carb> **S** | <Carburetor, r/w, carb> **A**

  - Operationsfolge

OPERATION	DECISION	RELEVANT GRAMMAR PARSE
<Carburetor, w, carb>	rejected	C_3: **S** ← <Carburetor, w, carb> .
<Carburetor, r, carb>	accepted	C_3: **S** ← <Carburetor, r , carb> . **A** P_3: **S** ← <Carburetor, r , carb> . **A**
Carburetor, w, carb	accepted	C_1: **S** ← <any, w, carb> . **A** C_3: **S** ← <Carburetor, r , carb> . **A** **A** ← <Carburetor, w, carb> . **A** P_1: **S** ← <any, w, carb> . **A** P_3: **S** ← <Carburetor, r, carb> . **A** **A** ← <Carburetor, w, carb> . **A**
<Verify, r, carb>	accepted	C_1: **S** ← <any, w, carb> . **A** **A** ← <Verify, r, carb> . **B** C_4: **S** ← <Verify, r, carb> . **A** P_1: **S** ← <any, w, carb> . **A** **A** ← <Verify, r, carb> . **B** P_4: **S** ← <Verify, r, carb> . **A**
<Verify, inc, prob_count>	accepted	P_2: **S** ← . **B** **B** ← . **A** . . . **A** ← <any, inc, prob_count> . . . .
<Verify, r, spec>	accepted	C_5: **S** ← <Verify, r, spec> . **A** P_5: **S** ← <Verify, r, spec> . **A**

Kooperative Transaktionshierachien (9)

- Beispiel (Forts.)
  - Operationsfolge (Forts.)

OPERATION	DECISION	RELEVANT GRAMMAR PARSE
<Verify, w, spec>	accepted	C_1: **S** ← <any, w, carb> . **A** **A** ← <Verify, r, carb> . **B** **B** ← <Verify, w, spec> . **S** C_5: **S** ← <Verify, r, spec> . **A** **A** ← <Verify, w, spec> . **A** P_1: **S** ← <any, w, carb> . **A** **A** ← <Verify, r, carb> . **B** **B** ← <Verify, w, spec> . **S** P_5: **S** ← <Verify, r, spec> . **A** **A** ← <Verify, w, spec> . **A**
<Carburetor, w, carb>	accepted	C_1: **S** ← <any, w, carb> . **A** **A** ← <Verify, r, carb> . **B** **B** ← <Verify, w, spec> . **S** **S** ← <any, w, carb> . **A** C_3: **S** ← <Carburetor, r , carb> . **A** **A** ← <Carburetor, w, carb> . **A** **A** ← <Carburetor, w, carb> . **A** C_4: **S** ← <Verify, r, carb> . **A** **A** ← <$\overline{\text{Verify}}$, w, carb> . **S** P_1: **S** ← <any, w, carb> . **A** **A** ← <Verify, r, carb> . **B** **B** ← <Verify, w, spec> . **S** **S** ← <any, w, carb> . **A** P_3: **S** ← <Carburetor, r, carb> . **A** **A** ← <Carburetor, w, carb> . **A** **A** ← <Carburetor, w, carb> . **A** P_4: **S** ← <Verify, r, carb> . **A** **A** ← <$\overline{\text{Verify}}$, w, carb> . **S**

Kooperative Transaktionshierachien (10)

- Beispiel (Forts.)
  - Operationsfolge (Forts.)

OPERATION	DECISION	RELEVANT GRAMMAR PARSE
<Carburetor, dec, prob_count>	accepted	P_2: **S** ← . **B** **B** ← . **A** . . . **A** ← <any, inc, prob_count> <any, dec, prob_count> .
<Verify, r, carb>	accepted	C_1: **S** ← <any, w, carb> . **A** **A** ← <Verify, r, carb> . **B** **B** ← <Verify, w, spec> . **S** **S** ← <any, w, carb> . **A** **A** ← <Verify, r, carb> . **B** C_4: **S** ← <Verify, r, carb> . **A** **A** ← <$\overline{\text{Verify}}$, w, carb> . **S** **S** ← <Verify, r, carb> . **A** P_1: **S** ← <any, w, carb> . **A** **A** ← <Verify, r, carb> . **B** **B** ← <Verify, w, spec> . **S** **S** ← <any, w, carb> . **A** **A** ← <Verify, r, carb> . **B** P_4: **S** ← <Verify, r, carb> . **A** **A** ← <$\overline{\text{Verify}}$, w, carb> . **S** **S** ← <Verify, r, carb> . **A**

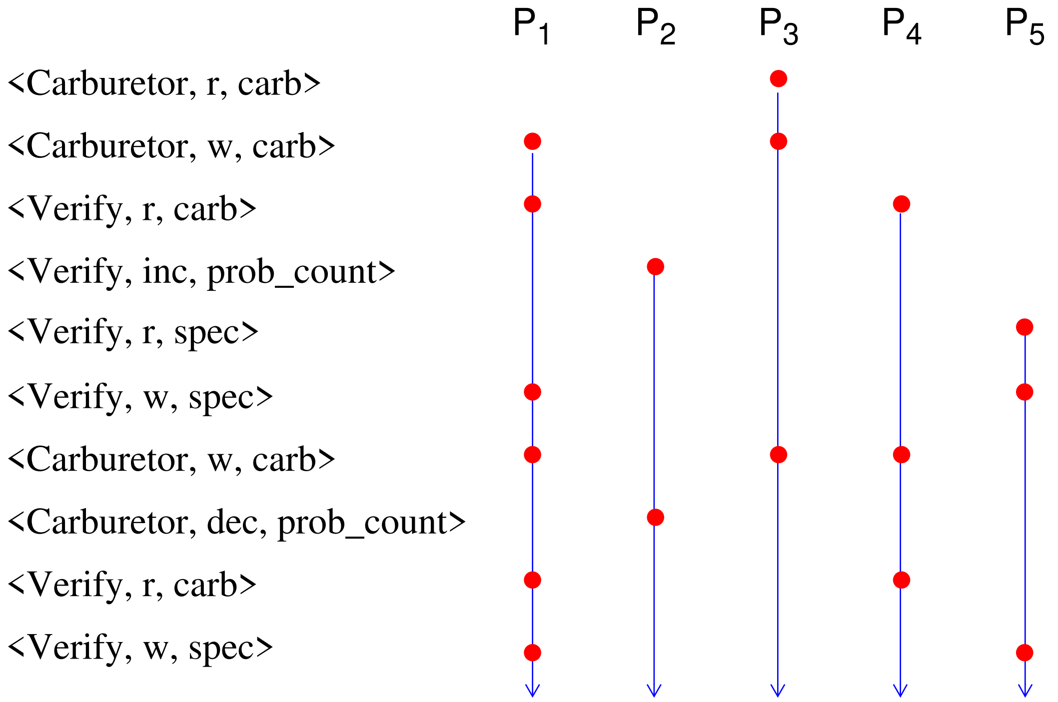
Kooperative Transaktionshierachien (11)

- Beispiel (Forts.)
  - Operationsfolge (Forts.)

OPERATION	DECISION	RELEVANT GRAMMAR PARSE
<Verify, w, spec>	accepted	C_1: **S** ← <any, w, carb> . **A** **A** ← <Verify, r, carb> . **B** **B** ← <Verify, w, spec> . **S** **S** ← <any, w, carb> . **A** **A** ← <Verify, r, carb> . **B** **B** ← <Verify, w, spec> . **S** C_5: **S** ← <Verify, r, spec> . **A** **A** ← <Verify, w, spec> . **A** **A** ← <Verify, r, spec> . **A** P_1: **S** ← <any, w, carb> . **A** **A** ← <Verify, r, carb> . **B** **B** ← <Verify, w, spec> . **S** **S** ← <any, w, carb> . **A** **A** ← <Verify, r, carb> . **B** **B** ← <Verify, w, spec> . **S** P_5: **S** ← <Verify, r, spec> . **A** **A** ← <Verify, w, spec> . **A** **A** ← <Verify, w, spec> . **A**

Kooperative Transaktionshierachien (12)

- Beispiel (Forts.)
  - Historien bzgl. jeder einzelnen Grammatik



  - Checkpoint von *Carburetor* und *Verify*

P_1: **S** ← <any, w, carb> **A** . **A** ← <Verify, r, carb> **B** . **B** ← <Verify, w, spec> **S** . **S** ← <any, w, carb> **A** . **A** ← <Verify, r, carb> **B** . **B** ← <Verify, w, spec> **S** . **S** ← ε . P_2: **S** ← **B** . **B** ← **A** . **A** ← <any, inc, prob_count> <any, dec, prob_count> . P_3: **S** ← <Carburetor, r, carb> **A** . **A** ← <Carburetor, w, carb> **A** . **A** ← <Carburetor, w, carb> **A** . **A** ← ε .	P_4: **S** ← <Verify, r, carb> **A** . **A** ← <Verify, w, carb> **S** . **S** ← <Verify, r, carb> **A** . **A** ← ε . P_5: **S** ← <Verify, r, spec> **A** . **A** ← <Verify, w, spec> **A** . **A** ← <Verify, w, spec> **A** . **A** ← ε .

Kooperative Transaktionshierachien (13)

❑ Beispiel (Forts.)

➪ Recovery

- Abhängigkeiten zwischen Operationen

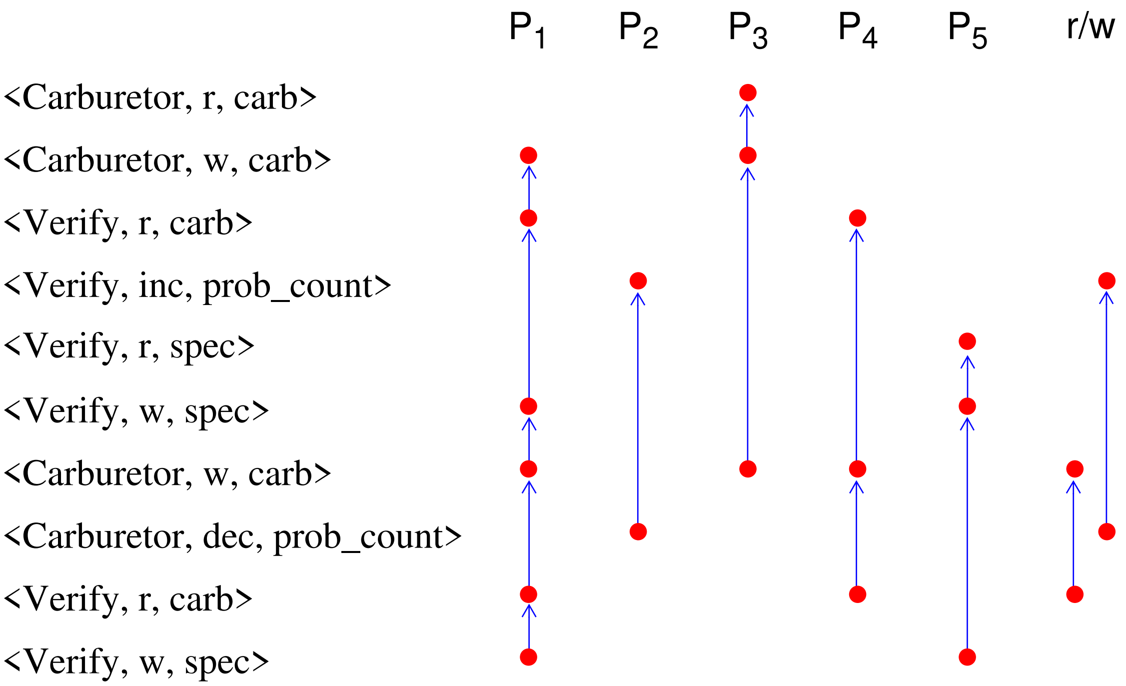
Historie von TG Engine	Historie von TG Car Design
<Carburetor, r, carb>	<Engine, r, carb>
<Carburetor, w, carb>	
<Verify, r, carb>	
<Verify, inc, prob_count>	<Engine, r, prob_count>
<Verify, r, spec>	<Engine, r, spec>
<Verify, w, spec>	
<Carburetor, w, carb>	
<Carburetor, dec, prob_count>	
<Verify, r, carb>	
<Verify, w, spec>	
Carburetor and Verify checkpoint	<Engine, w, carb>
	<Engine, w, spec>

Kooperative Transaktionshierachien (14)

❑ Beispiel (Forts.)

➯ Recovery (Forts.)

- Invalidierungsphase umfasst jede Operation, für die gilt:
  - es wurde explizit Abort ‘angeordnet’
  - es besteht eine Pattern-, Reads-From-, oder Parent-Child-Abhängigkeit zu einer Operation, die invalidiert wurde
  - die Operation würde bzgl. der neuen (bereinigten) Historie einen Konflikt (bzgl. einer gegebenen Confict-Grammatik) hervorrufen
- Beispiel



- nach Invalidierung Benachrichtigung der betroffenen Teilnehmer und Durchführung der eigentlichen Recovery-Aktionen durch die betroffenen Teilnehmer

Kooperative Transaktionshierachien (15)

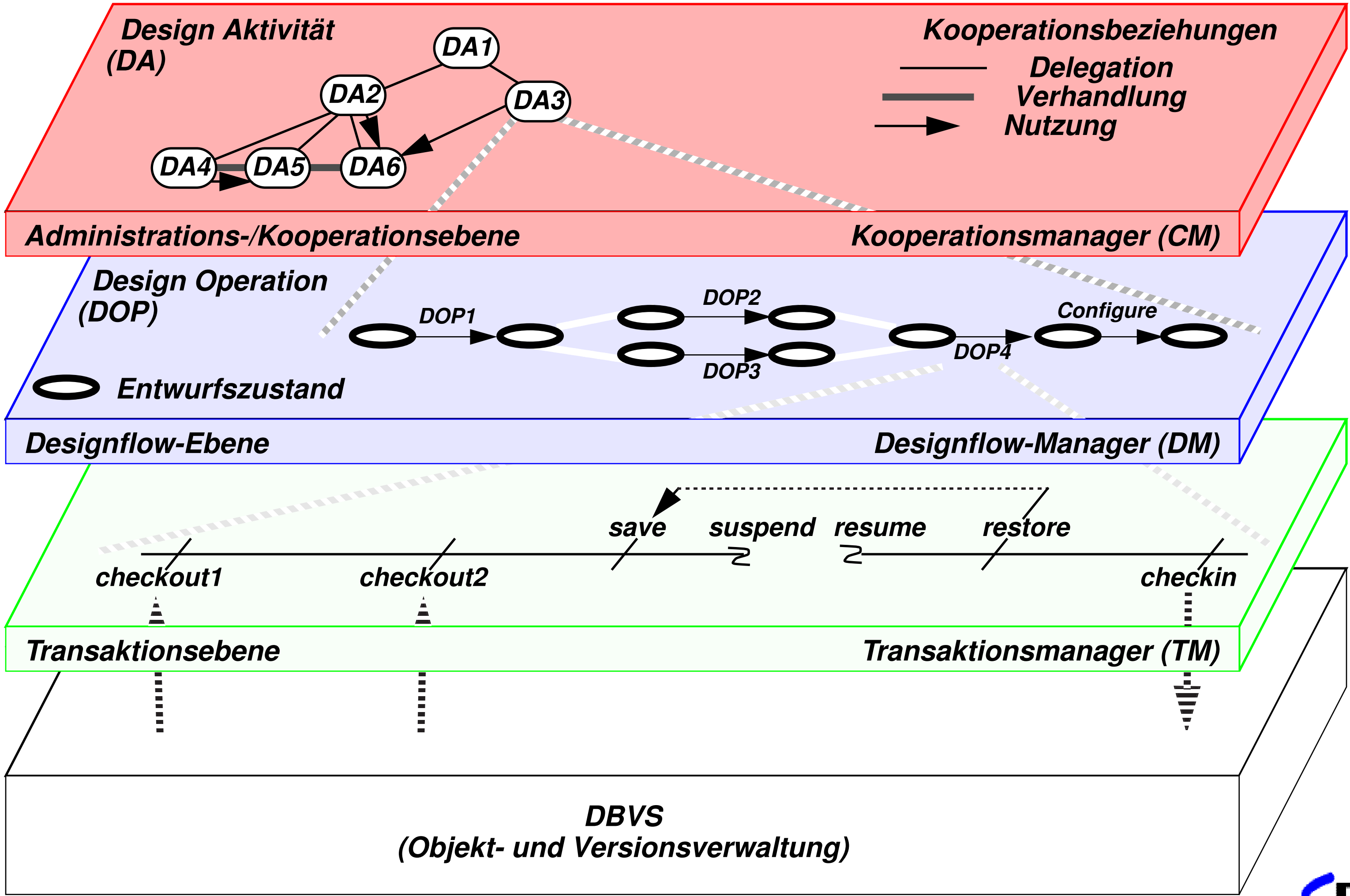
- Zusammenfasssung
  - Hierarchisierung
  - benutzerdefinierte Korrektheit
  - abgestufte Recovery
  - implizite Versionierung

- Problem
  - unverhältnismäßig hoher Spezifikationsaufwand
  - Unüberschaubarkeit der spezifizierten Korrektheitskriterien
  - unnatürliche ‘Ebene’ der Spezifikation von Kooperationsmustern
  - keine Berücksichtigung expliziter Versionierung

- Neuer Ansatz
  - Basis: angemessenes Objekt- und Versionsmodell
  - explizite (systemseitige) Repräsentation ‘natürlicher’ Ablaufeinheiten
  - Bereitstellung spezifischer Kontrollmechanismen auf jeder ‘Ebene’
    - Kontrolle der Entwurfsdatenverarbeitung (Entwurfstransaktionen)
    - Koordination von Entwurfswerkzeuganwendungen (Designflows)
    - Kontrolle der Kooperation zwischen Entwurfsaufträgen/ Entwerfern (‘Sharing’, Zugriffsrechte)

CONCORD(1): Überblick

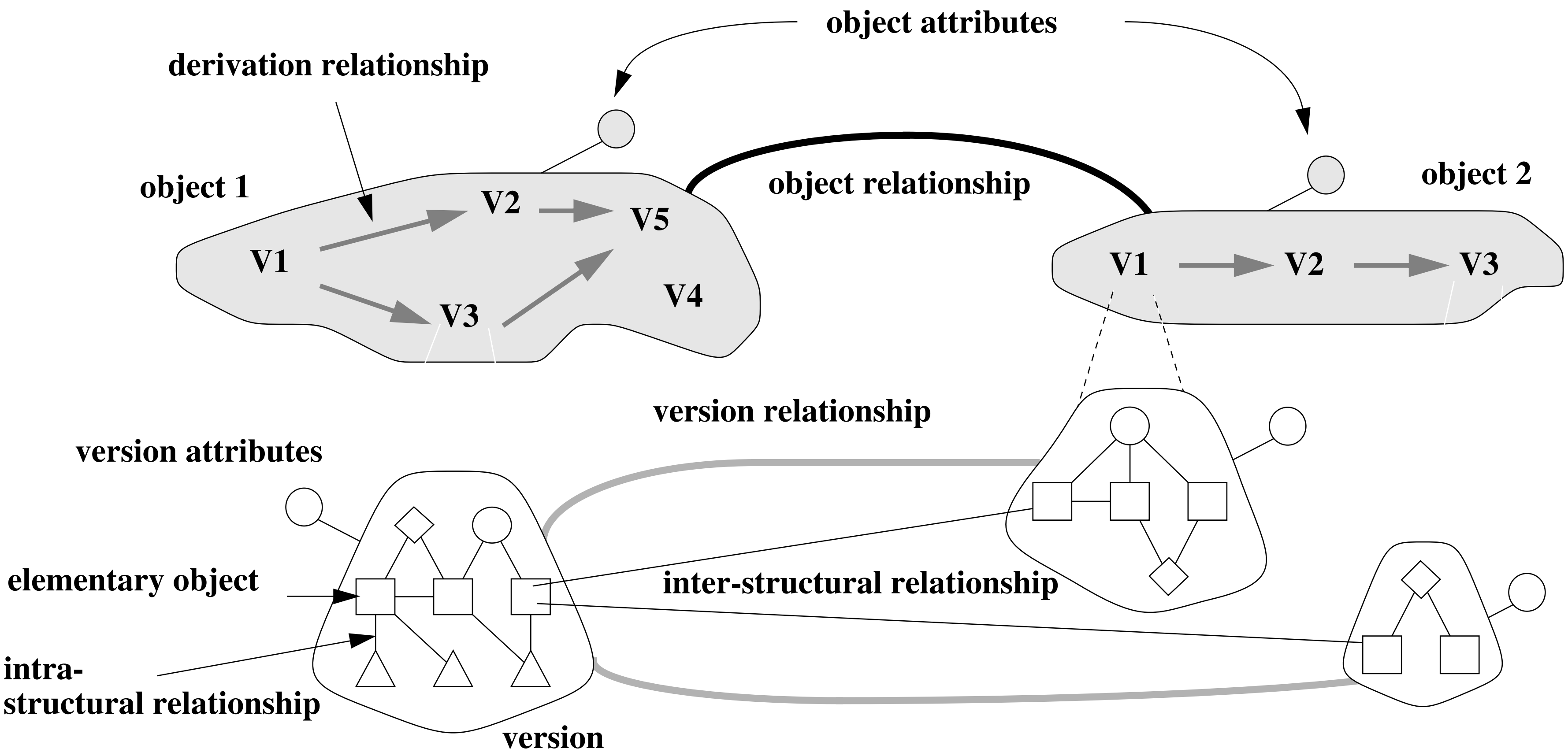
(CONtrolling COopeRation in Design Environments)



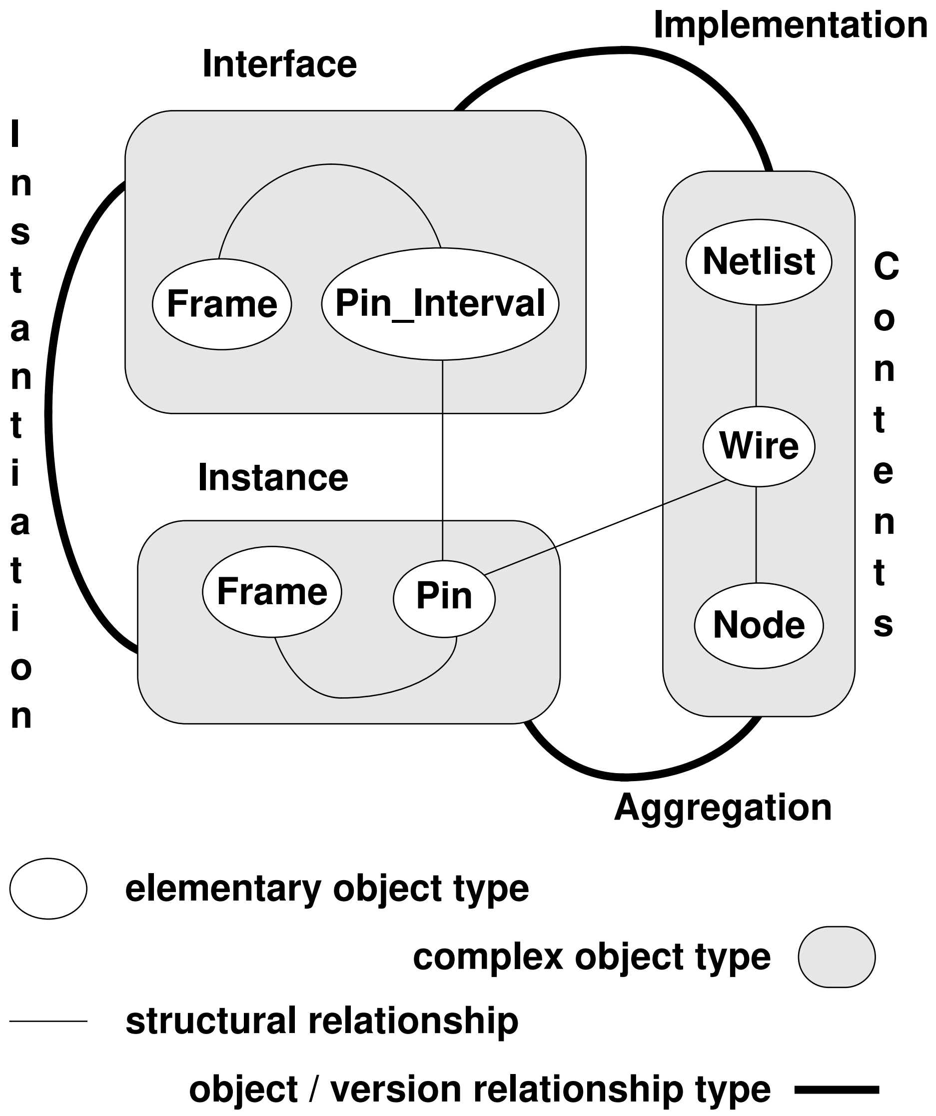
CONCORD(2): Objekt- und Versionsverwaltung (1)

- *Anforderungen*
  - Objektzustände komplexer Objekte
  - Abstraktion
  - Zerlegung und Konfigurierung
  - deskriptive, mengenorientierte Sprache
- *Prinzip*
  - allgemeines Versionsmodell: erleichtert Einbringen von Anwendungssemantik
  - explizite Versionierung/Konfigurierung
- *Beitrag zum 'Concurrent Engineering'*
  - natürliche Konfliktvermeidung
  - unterstützt 'schrittweise Verbesserung'
  - ermöglicht Zurückschreiten zu früheren Entwurfs(objekt)zuständen
  - Objekte, Versionen, Konfigurationen bilden Gegenstand der Kooperation

CONCORD(3): Objekt- und Versionsverwaltung (2)



CONCORD(4): Objekt- und Versionsverwaltung (3)



- **DDL:**

DEFINE OBJECT_TYPE Interface
  COMPONENTS (Frame, Pin_Interval)
  OBJECT_ATTRS (ifa_id: IDENTIFIER,
  name: STRING,
  function: STRING)
  VERSION_ATTRS (ifa_v_id: IDENTIFIER,
  scale_factor: REAL,
  no_feedthroughs: INTEGER)
  VERSION DERIVATION GRAPH IS TREE;

DEFINE VERSIONED OBJECT_TYPE Contents
  COMPONENTS (Netlist, Wire, Node)
  OBJECT_ATTRS (Content_Id : IDENTIFIER,
  Module_Name : STRING,
  Function : STRING)
  VERSION_ATTRS (Creator : STRING,
  Critical_Path : INTEGER)
DERIVATION IS TREE;

DEFINE LINK_TYPE Aggregation
  BETWEEN Contents **AND** Instance [1, *], [0, 1]
  FOR VERSIONS [1,*], [*, *];

CONCORD(5): Objekt- und Versionsverwaltung (4)

- DML

```
CREATE OBJECT
FROM Contents
(OBJECT_ATTRS          Module_Name := '1 Bit Adder',
                       Function := 'c:= b1 AND b2; s:= b1 XOR b2');

CREATE VERSION
FROM Contents
    VERSION_ATTRIBUTES   Creator := 'Mr. Ram',
                         Critical_Path := 25)
    PREDECESSOR          (No = 12)
    COMPONENTS           Netlist (77),
                         Wire (   id := #1,
                                  position := ...),
                         Wire (1002), …, Wire (1046),
                         Node (   id := #2,
                                  position := [(4.7,1.1),(0.8,1.5)],
                                  LINKED_TO Wire (#1,1046)),
                         Node (133), …
    LINKS                Implementation LINK TO Interface.Version(8)
WHERE Content.Content_Id = 1322;
```

CONCORD(6): Objekt- und Versionsverwaltung (5)

- DML (Forts.)

```
SELECT OBJECT
FROM      Contents - <Aggregation> Instance
WHERE     Module_Name = '1 Bit Adder';

SELECT VERSION
FROM      Interface - <Implementation> Contents
WHERE     EXISTS Interface: Frame (area < 50)
    AND   COUNT
          Contents    (Module_Name = '1-Bit-Adder') AND (Critical_Path < 30) > 2;
```

CONCORD(7): Objekt- und Versionsverwaltung (6)

- Konfigurationen (Forts.)

flach, homogen:

DEFINE CONFIGURATION_TYPE Cell1 **FOR** Instance
ATTRIBUTES
Cell_Id**:** **IDENTIFIER,**
Name**:** **STRING**
VIA Path1**:** Instance-<Instantiation>Interface-<Implementation>Contents [1,1]
OBJECT_SELECTION;

flach, erlaubt heterogene Ausprägungen:

DEFINE CONFIGURATION_TYPE Cell2 **FOR** Contents
ATTRIBUTES
Cell_Id**:** **IDENTIFIER,**
Name**:** **STRING**
VIA Path1**:** Contents-<Implementation>Interface-<Instantiation>Supercell(Instance) [1,1]
VIA Path2**:** Contents-<Aggregation>Subcell(Instance) **VERSION_SELECTION;**

CONCORD(8): Objekt- und Versionsverwaltung (7)

- Konfigurationen (Forts.)

rekursiv:

DEFINE CONFIGURATION_TYPE Cell3 **FOR** Instance
ATTRIBUTES
Cell_Id**:** **IDENTIFIER,**
Name**:** **STRING**
VIA Path1**:** Supercell(Instance)-<Instantiation>Interface-<Implementation>Contents [1,1]
(VIA Path2:Contents-<Aggregation>-Subcell(Instance))
FOR Subcell **USE** Cell3**;**

CONCORD(9): Objekt- und Versionsverwaltung (8)

- DML (Forts.) / Weitere Anweisungen
  - Selektieren von Konfigurationen (**SELECT CONFIGURATION**)
  - (schrittweises) Anlegen von Konfigurationen (**CREATE CONFIGURATION**)
    - explizite Auswahl
    - Spezifikation der Versionen über Identifikatoren oder Prädikate
  - Ändern/Löschen von Versionen (**UPDATE / DELETE VERSION**)
  - Kopieren von Versionen (**COPY VERSION**)
  - Änderung der Zugriffsmöglichkeiten auf Versionen (Objekte)
    - SET ACCESS MODE (READ, (ONE_)DERIVATION(_AT_A_TIME), UPDATE)
    - SET MAX_NUMBER OF SUCCESSORS
    - SET MAX_NUMBER OF ROOTS

CONCORD(10): Transaktionsebene (1)

- *Anforderungen*
  - Mittel zur Beschreibung/Programmierung von Design-Operationen
  - transaktionsgeschützte Ausführung ...
  - ... aber Konfliktbehandlung ‘im Sinne des *Concurrent Engineering*’ !

- *Mechanismen und Beiträge zur Ablaufkontrolle*
  - Programmierschnittstelle ‘unterstützt’ Verarbeitungsmodell und Versionsmodell
  - Synchronisation: C^3-Sperrprotokoll
    - Grundlage: striktes 2-Phasen-Sperrprotokoll
    - Sperrmodi und Kompatibilitäten reflektieren Prinzipien des Versionsmodells
    - frühe Konflikterkennung und -auflösung (kein ‘Blocking’)
  - Erhaltung der semantischen Integrität
    - Bedingungen formulierbar auf den Strukturen des Versionsmodells
    - Unterstützung von Integritätsprüfungen auf Server und Workstation
    - flexible Ausnahmebehandlungsmöglichkeiten (‘Abort nur im Notfall’)

CONCORD(11): Transaktionsebene (2) - API

❑ **Struktur**

	Begin_of_DOP
Checkout	**Evaluate (Query$_1$)** ⋮ **Evaluate (Query$_n$)**
Datenmanipulationsphase	**[** **Änderungen, Versionsableitungen** **Save/Restore** **Suspend/Resume** **Evaluate (Query)** **Abort_DOP** **]**
Checkin	**End_of_DOP**

CONCORD(12): Transaktionsebene (3) - API

- **Beispiel**

```
SELECT   VERSION cell
         FOR DERIVATION
FROM     Instance-
         <Instantiation>Interface-
         <Implementation>Contents
WHERE    EXISTS Interface
                (no_feedthroughs < 3);
```

```
cellAbsInterfaceCursor *aifc;

DOP adop = DOP::begin();
q_cell->evaluate();
q_cell->Instance()->moveFirst();

do {
   aifc = q_cell->Interface()->moveFirst();
   cout << “Changing:” << aifc->id() << endl;
   aifc->pinInterval()->moveFirst()->x(newx);
   if (!correct(q_cell->Instance())) {
      q_cell->Instance()->createSuccessor();
      q_cell->Instance()->moveSuccessor();
      q_cell->Instance()->pin()->x(calcX(aifc));
      q_cell->Instance()->connect(aifc); }
} while(q_cell->Instance()->moveNext());
adop->end();
```

CONCORD(13): Transaktionsebene (4) - API

- Verarbeitungsprinzipien
  - ➪ einen Cursor-Bereich pro Checkout-Schritt
  - ➪ 'durchschlagende' Versionsmanipulation/-ableitung

- Transaktionale Operatoren
  - ➪ *Begin_Of_DOP, Save/Restore, Suspend/Resume*
  - ➪ *Abort_DOP*
    - Objektpufferinhalte 'vergessen'
    - Protokolldaten löschen
    - Sperren freigeben
    - DOP-Identifikator aus der Transaktionstabelle nehmen
  - ➪ *End_Of_DOP*
    - Bestimmen der Objektpuffer-Änderungsinformation
    - vorläufige Integration
    - Prüfen der Schemakonsistenz
    - bei Nichterfüllung Ausnahmebehandlung durchführen; falls weiterhin erfolglos, Integration zurücknehnmen
    - Ausführen der drei letzten zu ABORT_DOP beschriebenen Schritte

CONCORD(14): Transaktionsebene (5) - Synchronisation

- Sperrmodi auf Versionsebene

Version Lock Mode	Internal Effect	External Effect
VS (version read)	reading the version;	reading the version, deriving a successor version;
VD (version derive)	reading the version, deriving a successor version;	reading the version, deriving a successor version;
VDX (version exclusive derive)	reading the version, deriving a successor version;	reading the version;
VX (version exclusive)	reading the version, deriving a successor version, updating the version;	no rights;
VXC (version exclusive conv.)	reading the version;	no (further) rights;

CONCORD(15): Transaktionsebene (6) - Synchronisation

❑ **Kompatibilitäten auf Versionsebene**

lock held by a concurrent transaction (columns: VS = Version Read, VD = Version Derive, VDX = Version Derive Exclusive, VXC = Version Exclusive Convertible, VX = Version Exclusive)

lock requested (rows)

	lock requested	VS	VD	VDX	VXC	VX
SELECT VERSION [FOR READ]	VS	+	+	+	-	-
SELECT VERSION FOR DERIVATION	VD	+	+/-	*	-	-
	VDX	+	*	-	-	-
SELECT VERSION FOR UPDATE	VXC	+	+	+	-	-
	VX	-	-	-	-	-

CONCORD(16): Transaktionsebene (7) - Synchronisation

- Sperrmodi auf Objektebene

Object Lock Mode	Internal Effect	External Effect
OS (object shared)	reading object attributes, version manipulations;	reading object attributes, version manipulations;
OD (object derive)	reading object attributes, reading versions, inserting a root-version;	reading object attributes, version manipulations, (inserting a root-version);
OX (object exclusive)	exclusive access;	no rights;
OXC (object update conv.)	reading object attributes, manipulating versions;	no (further) rights;

CONCORD(17): Transaktionsebene (8) - Synchronisation

❑ **Kompatibilitäten auf Objektebene**

lock held by a concurrent transaction

lock requested

		OS	OD	OD	OXC	OX
SELECT OBJECT [FOR READ]	OS	+	+	+	-	-
SELECT OBJECT FOR DERIVATION	OD	+	+	+/-	-	-
SELECT OBJECT FOR UPDATE	OXC	+	+	+	-	-
	OX	-	-	-	-	-

CONCORD(18): Transaktionsebene (9) - Synchronisation

- Prinzipien
  - Vorabspezifikation des Zugriffsmodus
  - ‘Sperrverhandlungen’
  - Upgrade/Downgrade von Sperrmodi
  - Ausnutzung von Heuristiken (Konversion, Eskalation)

- Regeln
  - Kompatibilität muß für jeden Knoten der in der Anfrageergebnismenge enthaltenen Strukturen geprüft werden
  - Beispiel: SELECT VERSION FOR DERIVATION
    - erlauben DA-Hierarchie und Kooperationsbeziehungen den (ableitenden) Zugriff ?
    - muß VD oder VDX angefragt werden ?
    - besteht Kompatibilität ?
  - Eigentliche Modi für Versionsmanipulationen sind Paare:
    OS/VS; OS/VD; OS/VDX; OS/VX; OS/VXC.

CONCORD(19): Transaktionsebene (10) - Synchronisation

- Deadlocks
  - nur, wenn DOPs 'sich entscheiden' synchron zu warten
  - frühe Erkennung (während Checkout-Phase) und Behandlung (*Abort_DOP*)

- Korrektheit
  - Isolation Levels

0°	no Dependencies	Chaos
1°	Write-Write Dependencies	no Lost Update
2°	Write-Read Dependencies	no Dirty Read
3°	Read-Write Dependencies	Repeatable Read

  - SQL 2: Isolation Levels

0°	Browse
1°	Cursor Stability
2°	Repeatable Read (Serializability without Phantom Protection)
3°	Serializable

  - C^3 sichert SQL2-2°, d. h. 2.999° Isolation zu.
  - Braucht man Phantom-Schutz in Entwurfsanwendungen ?

CONCORD(20): Transaktionsebene (11) - Semantische Integrität

- **Integritätsbedingungen auf Objekt-/Versionsstrukturen**

DEFINE	**STATIC ASSERTION** IB1 **ON VERSIONS**
SCOPE	**GLOBAL**
FOR CS	Interface - <Implementation> Contents
WHERE	(area > 100)
AS	**FOR_ALL CS** Interface**: COUNT** Contents () **<** 8)
MODE	**DEFERRED;**

CONCORD(21): Transaktionsebene (12) - Semantische Integrität

- **Integritätsbedingungen auf Konfigurationen**

```
DEFINE    ASSERTION Minimal_2bitadder_Cfg ON CONFIGURATIONS
SCOPE     GLOBAL
FOR       CFG_TYPE Cell2
WHERE     Supercell.name = “2 Bit Adder”
AS        FOR_ALL CONF Cell2:
          (COUNT Subcell (name = “1 Bit Adder”) = 2) AND
          (COUNT Subcell (name = “Half-Adder”) = 1)
MODE      DEFERRED;
```

CONCORD(22): Transaktionsebene (13) - Semantische Integrität

- **Übergangsbedingungen**

DEFINE	**TRANSITIONAL ASSERTION** If_Trans_IC
	ON VERSIONS (OLD REFERS TO PREDECESSOR)
SCOPE	**GLOBAL**
FOR CS	Interface
WHERE	Interface.name **=** “2 Bit Adder”
AS	**FOR_ALL OLD**(Interface)**:**Frame (**EXISTS NEW**(Interface)**:**Frame@NF
	(NF.area **<** area)
MODE	**DEFERRED**;

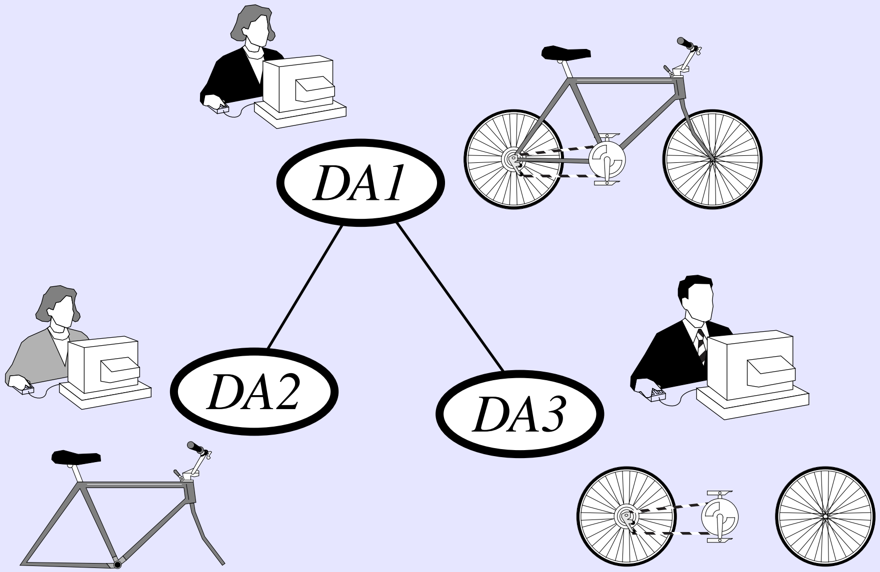
CONCORD(23): Transaktionsebene (14) - Semantische Integrität

- Integritätsprüfungen
  - Prüfung bei *‘Commit’* (MODE DEFERRED)
    - Reichweite: gesamte DB (SCOPE GLOBAL
      - alle TA (Schemakonsistenz)
    - Reichweite: auftragsspezifischer Datenbestand einer DA (SCOPE LOCAL)
      - alle TA der entsprechenden DA (auftragsspezifische Konsistenz)
    - bei Konflikten (Integritätsverletzungen):
      in die Commit-Behandlung integrierte Ausnahmebehandlung

CONCORD(24): Transaktionsebene (14) - Semantische Integrität

- Integritätsprüfungen (Forts.)
  - Prüfung während der Transaktion
    - kein IMMEDIATE
    - CHECK-Operation
      - explizite Einschränkung der Reichweite auf Objektpuffer-Datenbestand
      - Auswertung auf Gesamtdatenbestand
        - nur, wenn nicht auf weitere Sperren gewartet werden muß
        - in jedem Fall

CONCORD(25): Designflow-Ebene (1)

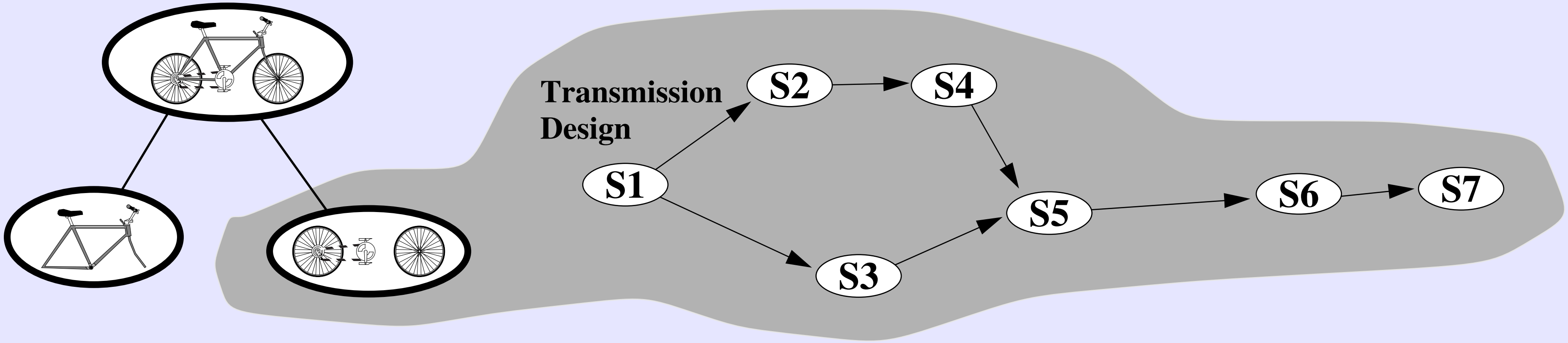


- *Erzeugung einer DA*
  - initialer Datenbereich
  - Entwurfsobjekt-Spezifikation
  - Entwurfsziel-Spezifikation
  - Designflow-Spezifikation
  - verantwortlicher Entwerfer

- *Anforderungen*
  - Koordination von Werkzeuganwendungen
  - sowohl geregelte als auch ungeregelte Phasen
  - Führung des Entwerfers ↗ | Automatisierung ↘

<u>*Assistenzsystem*</u>

CONCORD(26): Designflow-Ebene (2)



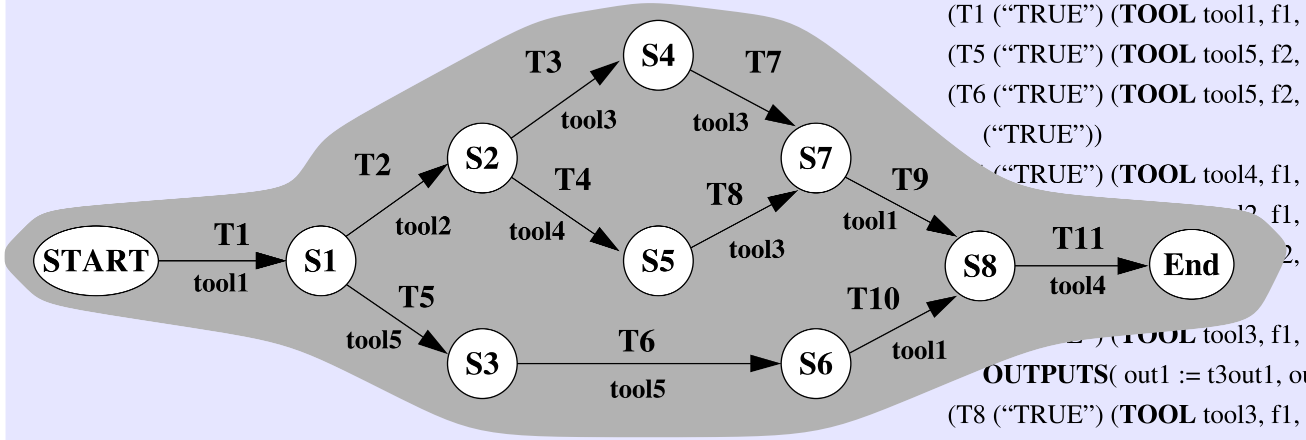
- *Vorplanung in einem CDFSL-Skript (unterstützt durch grafische Oberfläche)*
- *Basiskonstrukte*
  - Zustand: Menge zugreifbarer, auftragsspezifischer Entwurfsdaten
  - Transition: Vorbedingung, Entwurfsaktion, Nachbedingung
- *Typen von Entwurfsaktionen*
  - elementare Aktion: Entwurfswerkzeuganwendung
  - Sub-Designflow
  - ‘abstrakte’ Aktion
  - ‘komplexe’ Aktion

ACHTUNG: Im WS0102 gelten die Folien 42 bis 67 dieses Kapitels als (nicht-prüfungsrelevante) Ergänzung des eigentlichen Vorlesungsstoffes!

CONCORD(27): Designflow-Ebene (3)

DESIGNFLOW_DESCRIPTION SampleSimpleFlow
INPUT_PARAMETERS
(in1: OT1, in2: OT2, in3: **INTEGER**)
OUTPUT_PARAMETERS
(out1: OT2, out2: **INTEGER**)
OUTPUT_BINDINGS
(ou1 := End.t4out1, out2 := End.t4.out2)
TRANSITIONS (
(T1 (“TRUE”) (**TOOL** tool1, f1, **INPUTS**(t1in1 := Start.in1, t1in2 := Start.in3, t1in3 := 4) (“TRUE”))
(T5 (“TRUE”) (**TOOL** tool5, f2, **INPUTS**(t5in1 := S1.t1out2, t5in2 := S1.t1out1, t5in3 := 18) (“TRUE”))
(T6 (“TRUE”) (**TOOL** tool5, f2, **INPUTS**(t5in1 := S3.t5out1, t5in2 := S3.t5out2, t5in3 := S3.t5out3)
(“TRUE”))
“TRUE”) (**TOOL** tool4, f1, **INPUTS**(t4in1 := S2.out1, t4in2 := S1.t1out1) (“TRUE”))
f1, **INPUTS**(t2in1 := S1.t1out2, t2.in2 := Start.in3) (“TRUE”))
2, **INPUTS**(t3in1 := S2.t2out1, t3in2 := Start.in3, t3in3 := S2.t2out3)

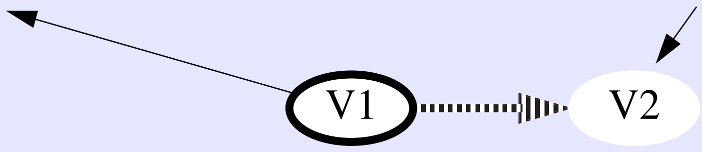
tool3, f1, **INPUTS**(t3in1 := S4.t3out1, t3in2 := S3.t3out2, t3in3 := S4.t3out3),
OUTPUTS(out1 := t3out1, out2 := t3out2)) (“TRUE”))
(T8 (“TRUE”) (**TOOL** tool3, f1, **INPUTS**(t3in1 := S5.t4out1, t3in2 := S5.t4out2, t3in3 := Start.in3),
OUTPUTS(out3 := t3out3)) (“TRUE”))
(T9 (“TRUE”) (**TOOL** tool1, f2, **INPUTS**(t1in1 := S7.out1, t1in2 := S7.out2, t1in3 := S7.out3),
OUTPUTS(s8out := t1out2)) (“TRUE”))
(T10 (“TRUE”) (**TOOL** tool1, f2, **INPUTS**(t1in1 := S6.t5out1, t1in2 := S6.t5out2, t1in3 := S6.t5out3),
OUTPUTS(s8out := t1out2)) (“TRUE”))
(T11 (“TRUE”) (**TOOL** tool4, f2, **INPUTS**(t4in1 := Start.in2, t4in2 := s8out) (“TRUE”)))
CONTROL_FLOW (
start->(T1, S1)->
ALTERNATIVES (
((T2, S2)->**PARALLEL**((T4, S5)->(T8, S7),(T3, S4)->(T7, S7))->(T9, S8)),
((T5, S3)->(T6, S6)->(T10, S8)))->
(T11, End));



ACHTUNG: Im WS0102 gelten die Folien 42 bis 67 dieses Kapitels als (nicht-prüfungsrelevante) Ergänzung des eigentlichen Vorlesungsstoffes!

CONCORD(28): Designflow-Ebene (4)

Tool1 (In: OT1, Func: FctSpec, PreCond: InIC, Out: OT1, PostCond: OutIC)



DEFINE	**TRANSITIONAL OV_ASSERTION** OutIC (Out: Identifier)
	ON VERSIONS (OLD REFERS TO PREDECESSOR)
SCOPE	**TOOL** Tool1
FOR CS	OT1
AS	**OLD**(Out.Attr) **<** **NEW**(Out.Attr)
MODE	**TOOL_POSTCONDITION;**

CONCORD(29): Designflow-Ebene (5)

(T6	(“	**DEFINE**	**PRE_CONDITION** T6PreCond (t5in1: Identifier, t5in2: Identifier)
		ON	**VERSIONS**
		FOR CS	OT1-OT2-OT3
		WHERE	(Id = t5in1) **AND EXISTS** OT2 (Id = t5in2)
		AS	...”)
	(**TOOL** tool5, f2,		**INPUTS** (t5in1 := S3.t5out1, t5in2 := S3.t5out2, t5in3 := S3.t5out3)
	(“	**DEFINE**	**POST_CONDITION** T6PostCond (t5out1: Identifier)
		ON	**VERSIONS**
		FOR CS	OT1-OT2-OT3
		WHERE	(Id = t5out1)
		AS	...”))

CONCORD(30): Designflow-Ebene (6)

- Modellierungselemente
  - Super-Designflow / Sub-Designflow: vgl. funktionaler Aspekt in MOBILE
    - Beispiel:

      (T6 (“**TRUE**”) (**SUB_DESIGNFLOW** SDF,
INPUTS (sdfin1 := S3.t5out1, sdfin2 := S3.t5out2)
(“**TRUE**”)).
  - Designflow-relevante Daten: vgl. informationsbezogenerAspekt in MOBILE
  - Kontrollfluß: vgl. verhaltensbezogenerAspekt in MOBILE
  - organisatorischer Aspekt
    - einer DA können mehrere Entwerfer zugeordnet werden
    - mehrere Entwerfer können gleichzeitig Sessions im Rahmen derselben DA durchführen, falls die Designflow-Spezifikation parallele Pfade beinhaltet

CONCORD(31): Designflow-Ebene (7)

- Modellierungselemente (Forts.)
  - abstrakte Aktivität
    - Beispiel:
      (T6 (“TRUE”) (**ABSTRACT_ACTIVITY** aaname) (“T6-Post-Condition”))
      ...
      ABSTRACT_ACTIVITY_DESCRIPTION aanme
      ACTIVITY_CONSTRAINTS (t1 **MUST_APPEAR**, t1 **LIMITS** t2)
    - vordefinierte, deskriptive Kontrollflußkonstrukte:
      A **LIMITS** B;
      A **ENABLES** B;
      A **IMPLIES** B;
      A **DIRECTLY_AFTER** B;
      A **AFTER** B;
      A **OR** B;
      A **MUST_APPEAR**;
      A **NOT**;
    - Definition weiterer Konstrukte über Spezifikation von Automaten (s. u.)

CONCORD(32): Designflow-Ebene (8)

- Modellierungselemente (Forts.)
  - komplexe Aktivität
    - ‘bedeutungsvolle’ Entwurfszustände:
      - der Zustand muß zu Spezifikationszwecken (z. B. Datenfluß) benannt werden
      - Zurückpositionieren des Designflows
      - Ansprechen von (vorl.) Entwurfsdaten zu Koperationszwecken (s. u.)
    - Nur der Anfangs- und Endzustand einer komplexen Aktivität ist ‘bedeutungsvoll’
    - Beispiel:
      (T6 (“TRUE”)
      (**ITERATION** (**SEQUENCE**
      TOOL tool5, f2, **INPUTS** (t5in1 := t1out1, t5in2 := 3, t5in3:= t1.ou2),
      TOOL tool1, f1, **INPUTS** (t1in1 := t5out1, t1in2 := t5out2, t1in3:=2)),
      INPUTS (t5in1 := S3.t5out1, t5in2 := S3.t5out2, t5in3 := S3.t5out3), 6)
      (“T6-Post-Condition”)).
    - Bei einer Iteration eines Sub-Designflows bleiben alle Zwischenzustände ‘bedeutungsvoll’

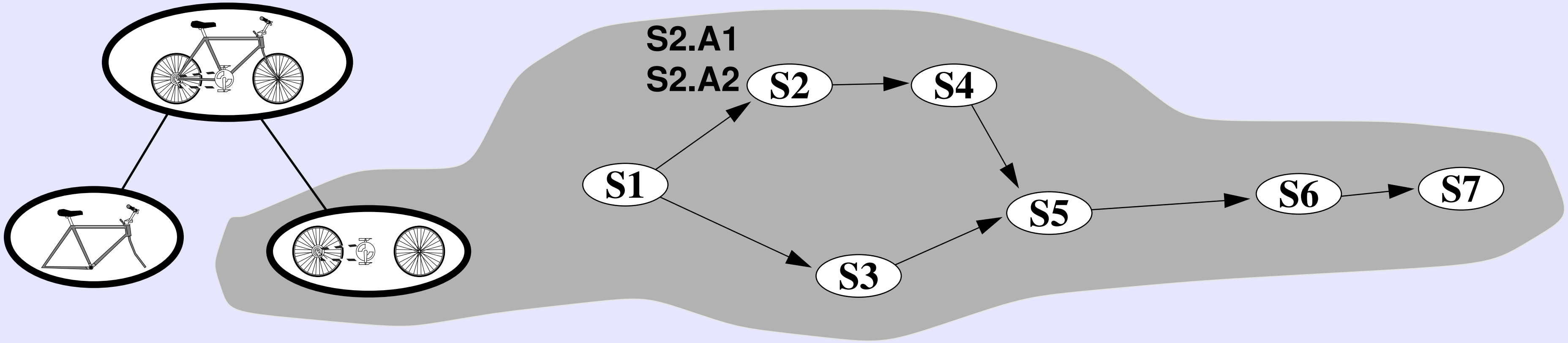
CONCORD(33): Designflow-Ebene (9)

- *Entwurfs(objekt)qualität*
  - ➪ Vor-, Nachbedingungen von Transitionen (s. o.)
  - ➪ vorgeplante oder dynamisch definierbare ‘Invarianten’
    - Beispiel: vorgeplante Invariante

      INTEGRITY_CONDITIONS (
      (S2**,** **“DEFINE** **INVARIANT** Invariante1
      ON **VERSIONS**
      FOR CS ...
      AS ...**”,**
      S8**)).**
    - Beispiel: dynamisch definierte, auftragsspezifische Integritätsbedingung

      DEFINE **CFG_ASSERTION** Hilfe1 **(**CfgId: Identifier**)**
      SCOPE **DA** DA1
      FOR CFG_TYPE Cell1
      WHERE Cell1.Id = CfgId
      AS ...
      MODE **ON_DEMAND;**
  - ➪ (durch das DBVS überprüfbare) Entwurfsziele (s. u.)

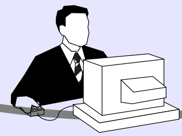
CONCORD(34): Designflow-Ebene (10)



- *Dynamische Ablaufentscheidungen*
  - Alternativverzweigungen
  - Datenfluß
  - abstrakte Phasen
- *Zurückschreiten im Designflow*
  - benutzerinitiiert, keine Kompensationstransaktionen
  - unterstützt durch History-Management

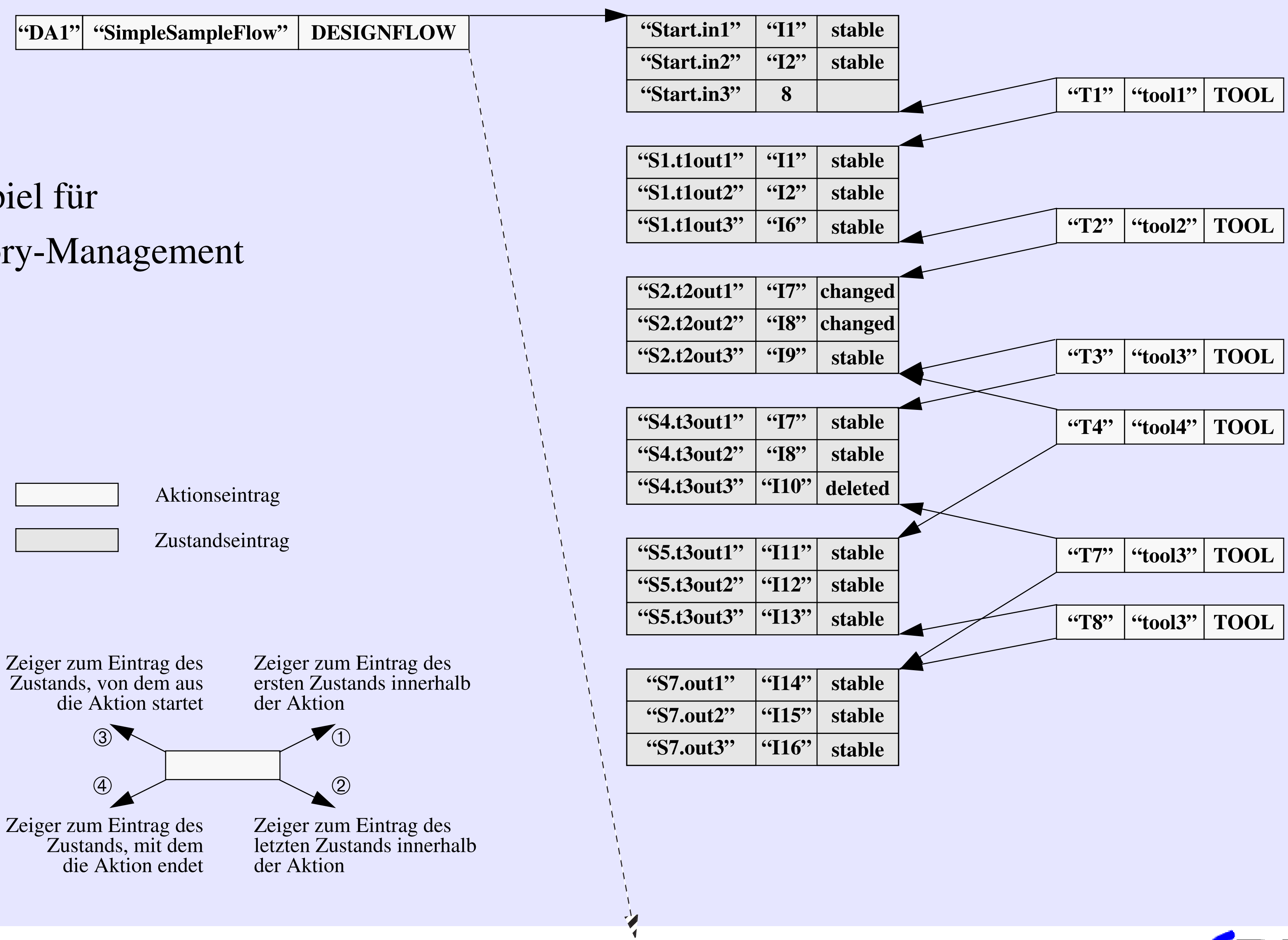
S1 → Tool1 → S2 → Tool2 → Tool3 → S4

u.U. Korrekturaktionen



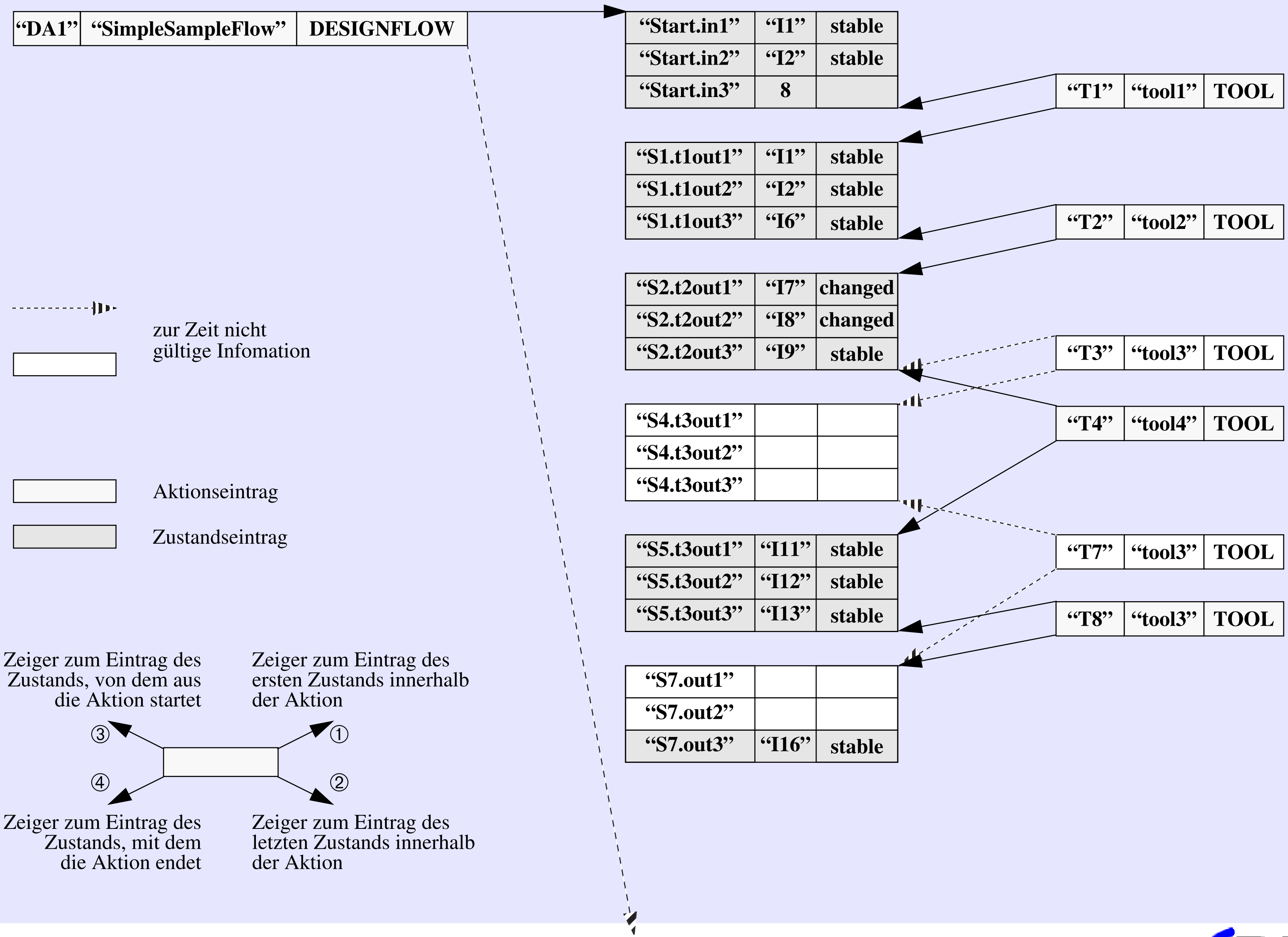
CONCORD(35): Designflow-Ebene (11)

Beispiel für
History-Management

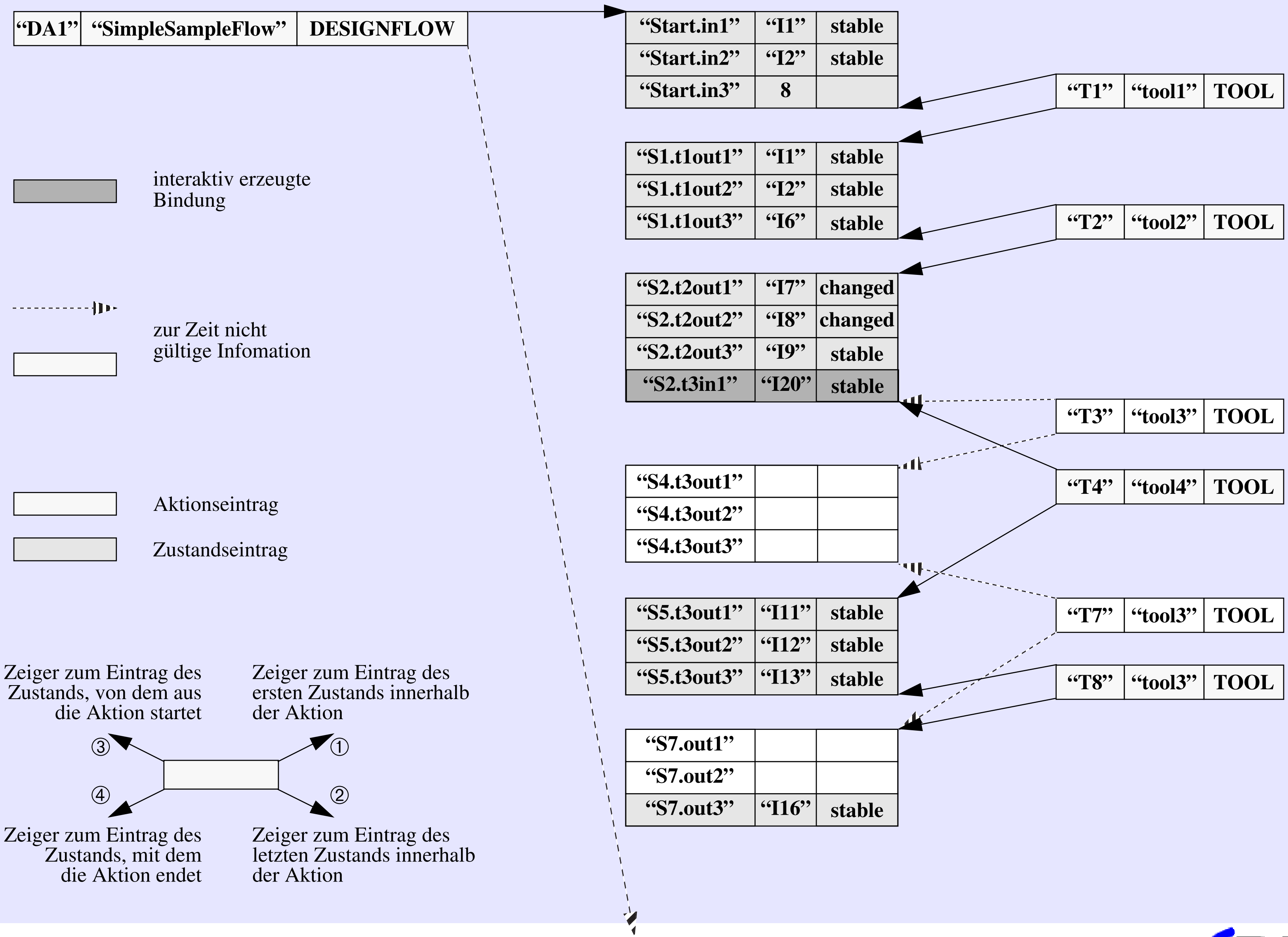


ACHTUNG: Im WS0102 gelten die Folien 42 bis 67 dieses Kapitels als (nicht-prüfungsrelevante) Ergänzung des eigentlichen Vorlesungsstoffes!

CONCORD(36): Designflow-Ebene (12)



CONCORD(37): Designflow-Ebene (13)



CONCORD(38): Kooperationsebene (1)

- *Anforderungen*
  - Auftragszerlegung und Beschreibung
    (von isolierter bis hin zu hochgradig kooperativer Auftragsbearbeitung)
  - Verwaltung von Datenbereichen / Kontrolle des ‘Information-Sharing’
    (von lokalen, auftragsbezogenen bis hin zu gemeinsamen Datenbereichen)
  - Grad der Vorplanung
    (von vollständiger Vorplanung bis hin zu dynamischen Abstimmungsaktionen)
- *Mechanismen*
  - gezielter ‘Austausch’ <u>bedeutungsvoller</u> Entwurfsobjektzustände
    - wertunabhängige Zugriffskontrolle
    - vorgeplante oder dynamische Ausführung von Kooperationsoperationen (*Request, Permit, Revoke, Transmit*)
  - vorgeplante Kooperation: *kooperative* Designflows
  - zusätzliche Kontrolle durch (auftragsspezifische) Integritätsbedingungen
  - automatische sowie benutzerveranlaßte Notifikation

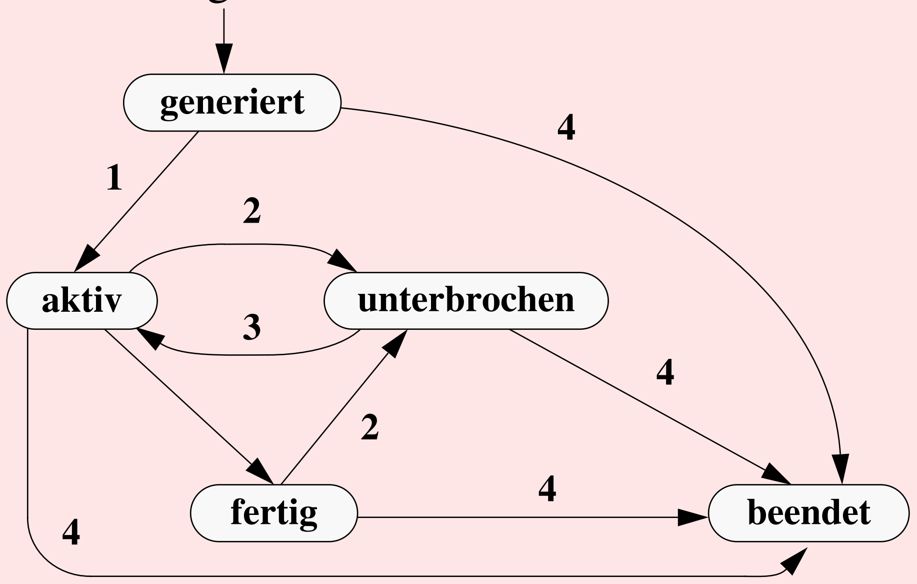
CONCORD(39): Kooperationsebene (2)

- *Delegation*
  - Erzeugen und Löschen von DAs
    - *Create_DA (<DA-Name>)*
    - *Transmit_Design_Data (<DA-Name-1>, <Design_Data_Specification>)*
      - *SCOPE(<DA-Name-2>)* (nur für Sub-DA)
      - OQL-SELECT-Anweisung (für Sub-DA oder Wurzel-DA)
      - *ALL* (nur Wurzel-DA)
    - *Associate_Design_Goal (<DA-Name>, <Formal-spezifizierte-Entwurfsziele>, <Informell-spezifizierte-Entwurfsziele>)*
      - *Beispiel:*

        DEFINE **DESIGN_GOAL** DA1Goal1
        FOR CFG_TYPE Cell1
        AS **EXISTS** Cell1 (...);
    - *Associate_Designflow (<DA-Name>, <Designflow-Specification-Name>)*
    - *Associate_Designer (<DA-Name>, <Designer-Name>)*

CONCORD(40): Kooperationsebene (3)

- *Delegation (Forts.)*
  - Beeinflussung von DAs zur Laufzeit



1 Start_DA
2 Suspend_DA
3 Resume_DA
4 Terminate_DA

- *Start_DA (<DA-Name>, <Cooperating-DAs>)*
- *Suspend_DA (<DA-Name>, <Mode>)*
  - *Mode: DIRECT | LAZY*
- *Resume_DA (<DA-Name>)*
- *Terminate_DA (<DA-Name>, <Data-Integration-Mode>))*
  - *Data-Integration-Mode: Complete_Integration | No_Integration*

CONCORD(41): Kooperationsebene (4)

- *Delegation (Forts.)*
  - Ändern der Vorgaben für eine DA
    - *Remove_Designflow, Associate_Designflow*
    - Verträglichkeitsprüfung durch Simulation der Historie in der neuen Spezifikation
      - Können die in der Historie enthaltenen Werkzeugläufe unter Beibehaltung der Ordnung auch in der neuen Spezifikation durchgeführt werden?
      - Assoziation des ermittelten (Designflow-)Zustands mit dem aktuellen Datenbestand der DA
      - Integritätsprüfungen

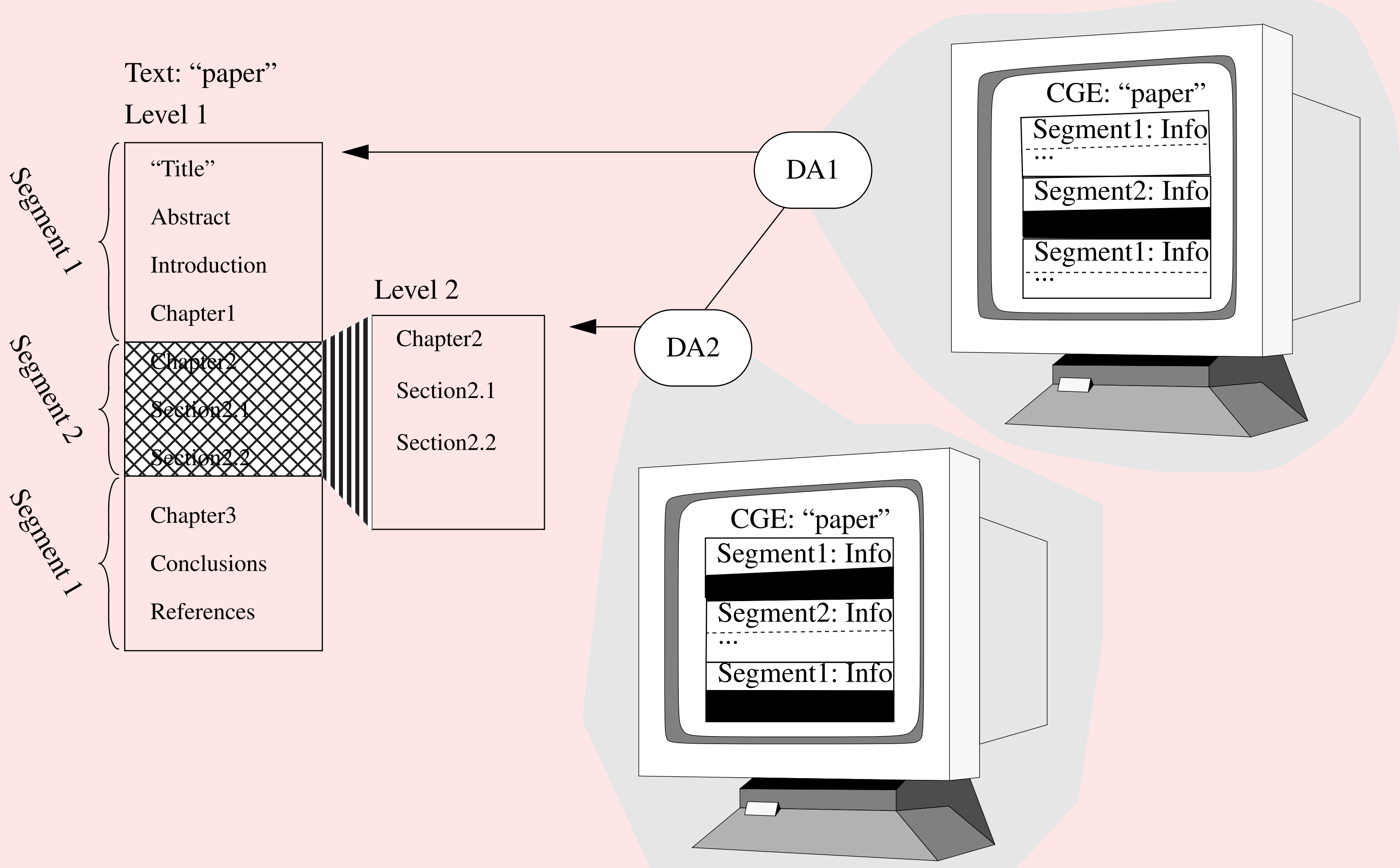
CONCORD(42): Kooperationsebene (5)

- *Nutzungsbeziehung*
  - Dynamische Kooperation
    - Vor.: Gegenseitige Kenntnis
      - Namen der zugeordneten Entwerfer
      - Entwurfszielspezifikation
      - Name der Designflow-Spezifikation
      - aktuelle Designflow-Zustände
    - Operationen
      - *Request (<DA-Name>, <Design-Data-Specification>, <Access-Mode>)*
        - *Design-Data-Specification*: Entwurfszustand, Entwurfsziele, OQL-SELECT
        - *Access-Mode*: ***READ, WRITE, DERIVE***
      - *Permit (<DA-Name>, <Design-Data-Specification>, <Access-Mode>)*
      - *Revoke (<DA-Name>, <Design-Data-Specification>, <Access-Mode>)*
      - *Transmit_Design_Data (<DA-Name>, <Design-Data-Specification>)*

ACHTUNG: Im WS0102 gelten die Folien 42 bis 67 dieses Kapitels als (nicht-prüfungsrelevante) Ergänzung des eigentlichen Vorlesungsstoffes!

CONCORD(43): Kooperationsebene (6)

- *Nutzungsbeziehung (Forts.)*
  - Anwendungsbeispiel für dynamische Kooperation: CGE



CONCORD(44): Kooperationsebene (6)

- *Nutzungsbeziehung (Forts.)*
  - Anwendungsbeispiel für dynamische Kooperation: CGE (Forts.)
    - Grundlagen / CONCORD-Prinzipien
      - transaktionsgeschützte Manipulation des Textes
      - manuelle Konfliktbehandlung im Falle von Zugriffskonflikten
      - Zuordnung von Textabschnitten (Segmenten) zu Personen
      - dynamische Kooperation
    - Enge Kooperation (innerhalb einer DA)
      - kurze Lesetransaktionen
      - Modifikation semantischer Einheiten (Markierung)
      - ‘Demand-Update’ der Frontend-Repräsentation
      - ‘Über die Schulter schauen’
    - Lose Kooperation (zwischen DAs)
      - explizite Zustandskennzeichnung von Textteilen
      - Nutzung der Kooperationsoperationen auf Segmenten

CONCORD(45): Kooperationsebene (6)

- *Nutzungsbeziehung (Forts.)*
  - Anwendungsbeispiel für dynamische Kooperation: CGE (Forts.)
    - Weitere Funktionen
      - Manipulation der DA-Hierarchie und der Textstruktur
      - Manipulation von Text
      - Benachrichtigungen und direkte Kommunikation
      - Annotation von Text
      - ‘einsehbare’ Historie *(Wer hat welche Textteile wann wie bearbeitet?)*

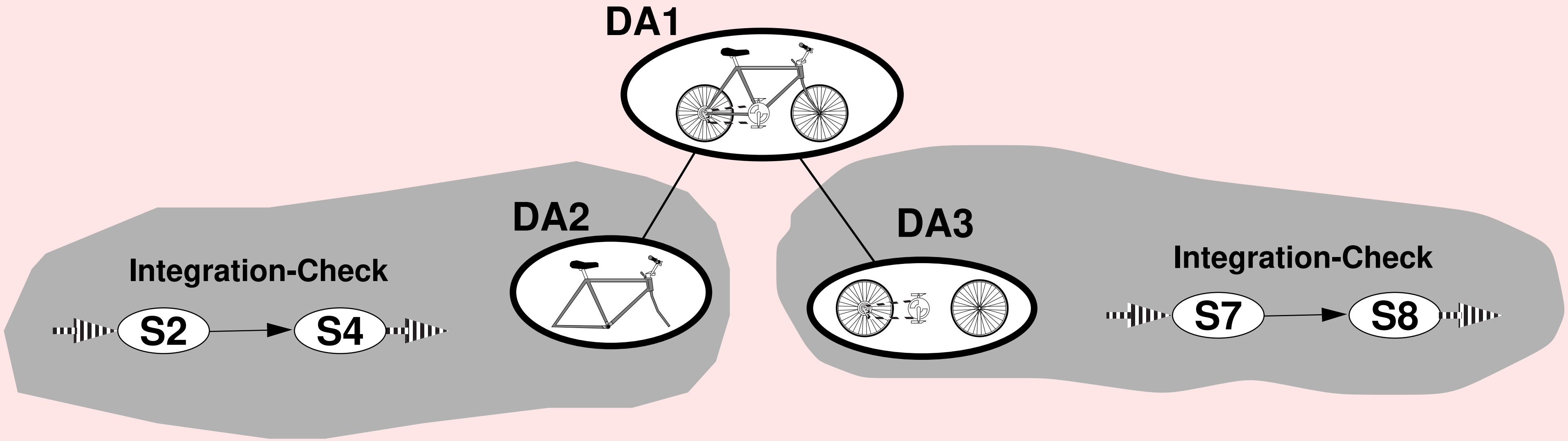
CONCORD(46): Kooperationsebene (7)

- *Nutzungsbeziehung (Forts.)*
  - Wartung von Nutzungsbeziehungen durch (automatische) Notifikation
    - wenn 'zur Verfügung gestellte' Daten geändert oder gelöscht worden sind (was jedoch in Anbetracht der Versionierung nur selten zu erwarten ist)
    - wenn die Rechte durch die besitzende DA wieder entzogen worden sind
    - bei semantischer Invalidierung (d. h., wenn die besitzende DA ihren Designflow über den Zustand, in dem die Rechte gewährt worden sind, zurückpositioniert)

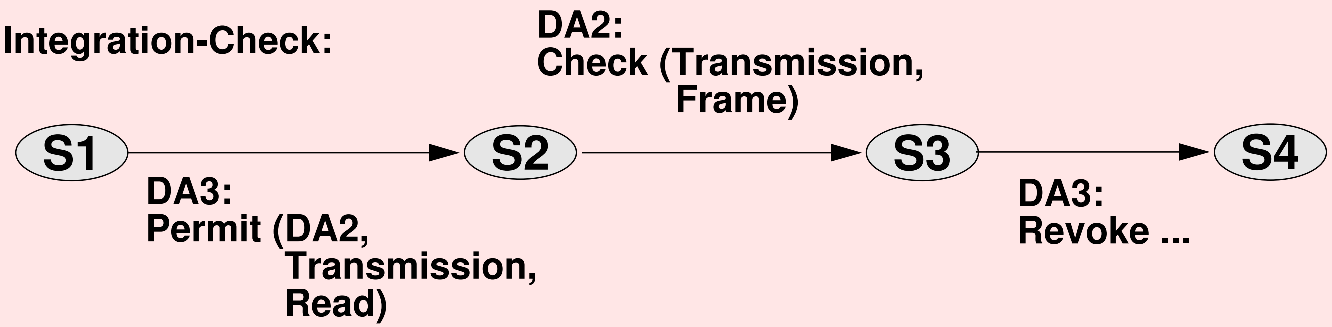
ACHTUNG: Im WS0102 gelten die Folien 42 bis 67 dieses Kapitels als (nicht-prüfungsrelevante) Ergänzung des eigentlichen Vorlesungsstoffes!

CONCORD(47): Kooperationsebene (8)

Beispiel: Vorgeplante Kooperation (*Kooperativer Designflow*)

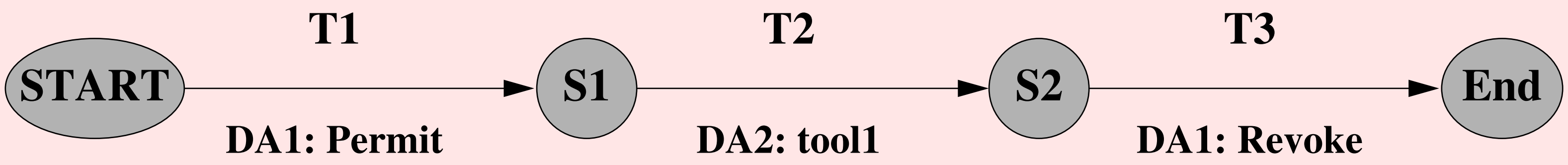


SUB-DESIGNFLOW Integration-Check:



CONCORD(48): Kooperationsebene (9)

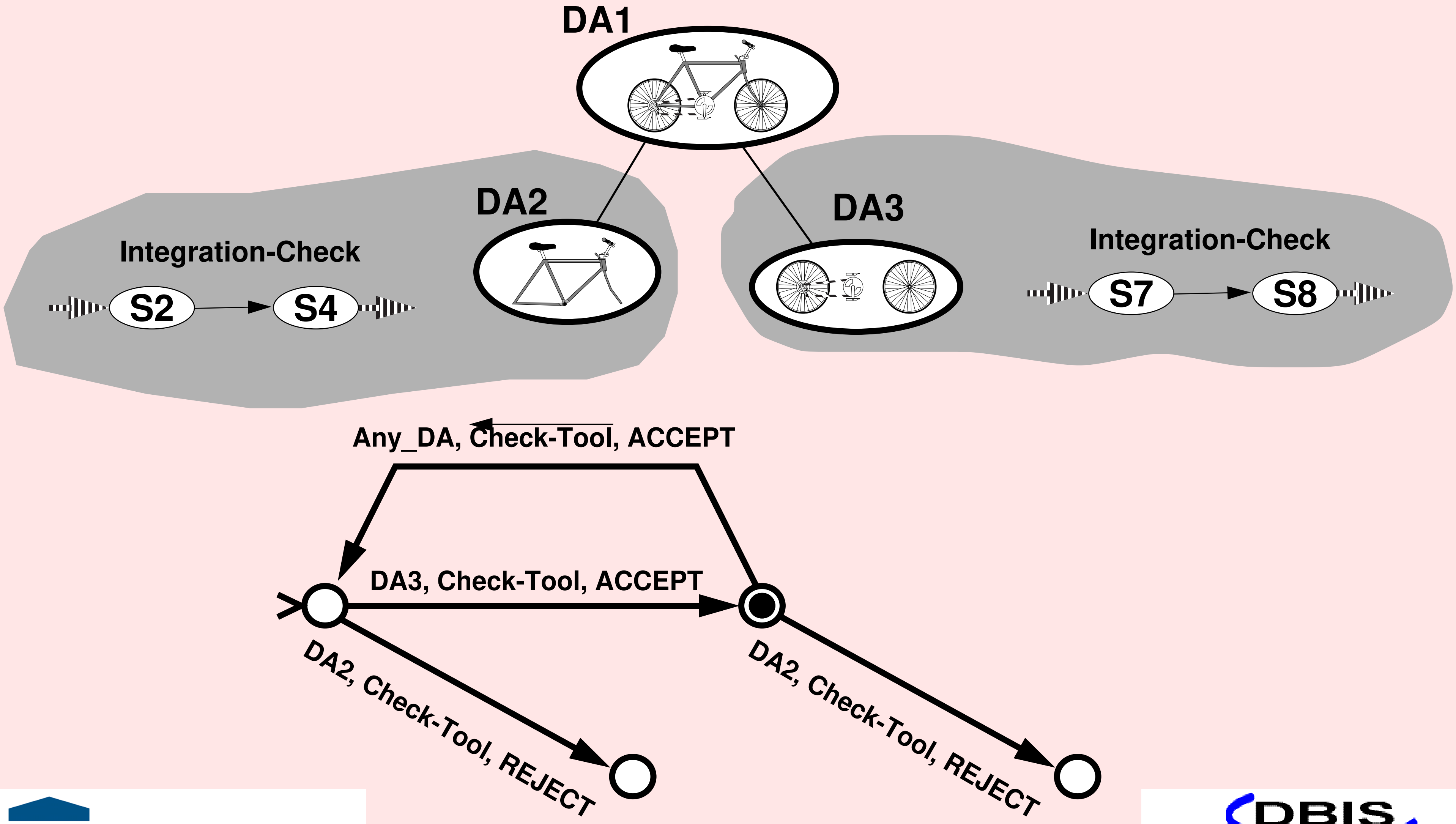
Beispiel: Vorgeplante Kooperation (Forts.)



COOPERATIVE_DESIGNFLOW_DESCRIPTION Abstimmung
INVOLVED ARE (DA1, DA2)
INPUT_PARAMETERS (in1: OT1,
in2: OT2)
OUTPUT_PARAMETERS (out1: OT2)
OUTPUT_BINDINGS (out1 := S2.t2out1)
TRANSITIONS (
(T1 (“TRUE”) (DA1: **PERMIT** (DA2, (**SELECT VERSIONS FROM** OT1 **WHERE** ID = in1), **DERIVE**))
(“TRUE”))
(T2 (“TRUE”) (DA2: **TOOL** (tool2), **INPUTS**(t2in1 := Start.in1, t2in2 := Start.in2) (“TRUE”))
(T3 (“TRUE”) (DA1: **REVOKE** (DA2, (**SELECT VERSIONS FROM** OT1 **WHERE** ID = in1), **DERIVE**))
(“TRUE”))
CONTROL_FLOW (
Start->(T1, S1)->(T2, S2)->(T3, End));

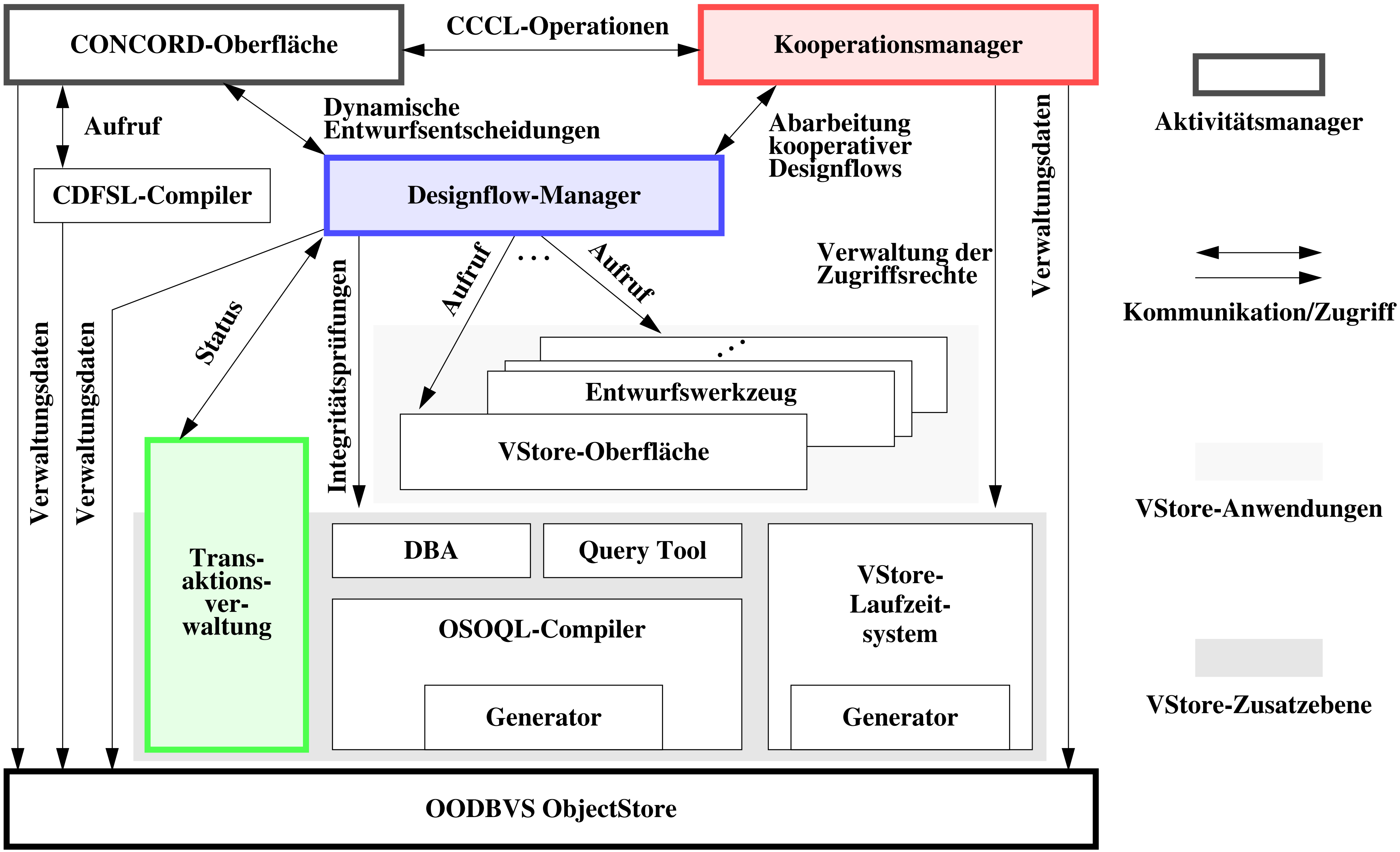
CONCORD(49): Kooperationsebene (10)

Beispiel: 'Nicht-präskriptive' Vorgaben bei (vorgeplanter) Kooperation



ACHTUNG: Im WS0102 gelten die Folien 42 bis 67 dieses Kapitels als (nicht-prüfungsrelevante) Ergänzung des eigentlichen Vorlesungsstoffes!

CONCORD(50): Architektur des CONCORD-Systems



CONCORD(51): Zusammenfassung - Designflow-Ebene

- *Nutzung von Designflows zur Vorgabe einer Entwurfsmethodik innerhalb eines Entwurfsauftrags (DA)*
- *Gestaltung von Designflows unter Anwendung der Prinzipien des Workflow-Managements*
  - Koordination von Werkzeuganwendungen
  - sowohl geregelte als auch ungeregelte Phasen (dynamische Ablaufentscheidungen)
  - Führung des Entwerfers ↘ | Automatisierung ↗
- *Dynamische Ablaufentscheidungen erfordern flexible(re) Fehlerbehandlungs-mechanismen*

CONCORD(52): Zusammenfassung - Kooperationsebene

- *Nutzung von Designflows und Mechanimsen der Zugriffskontrolle zur Unterstützung der Kooperation zwischen Entwurfsaufträgen (DAs)*
  - ➪ Delegationsbeziehung mit allgemeinem Protokoll
  - ➪ Nutzungsbeziehungen mit spezifischen Protokollen
- *Jede DA hat Zugriff auf*
  - ➪ private, auftragsspezifische Entwurfsdaten
  - ➪ öffentliche Entwurfsdaten
- *Kooperationsprimitive, wie Request, Permit, Revoke, Transmit*
  - ➪ kontrollieren den Austausch auftrgsspezifischer Daten
  - ➪ können in Designflows vorgegeben oder in bestimmten Phasen des Entwurfs dynamisch verwendet werden
- *Wartung von Kooperationsbeziehungen*
  - ➪ Notifikation

Zusammenfassung (kooperative Designflows in *CONCORD*)

- ***Explizites Versionsmodell***
  - ➪ angemessene Verwaltung von Entwurfsdaten(zuständen)
- ***Transaktionsebene***
  - ➪ transaktionsgeschützte Verarbeitung von versionierten Entwurfsdaten
  - ➪ Konfliktbehandlung im Sinne des Concurrent Engineering
- ***Designflow-Ebene***
  - ➪ Assistenzsystem (Verfolgen einer Entwurfsmethodik)
  - ➪ Zuordnung von ‘Entwurfsqualität’ zu Entwurfsobjektzuständen
- ***Kooperationsebene***
  - ➪ Assistenzsystem (Koordination von Entwerfertätigkeiten)
  - ➪ Kontrolle der Sichtbarkeit von Entwurfsdaten

CONCORD: ***Integrierte, datenorientierte Ablaufsteuerung für technische Entwurfsanwendungen!***