Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik
Institut für Technische Informationssysteme

Diplomarbeit

Föderierte Transaktionsverwaltung – eine vergleichende Betrachtung aktueller Entwicklungen

Verfasser:

Thomas Lehnecke

30.August 1996

Betreuer:

Dr. Stefan Conrad Dipl.-Inform. Can Türker

Universität Magdeburg
Fakultät für Informatik
Postfach 4120, D–39016 Magdeburg
Germany

Lehnecke, Thomas:
Föderierte Transaktionsverwaltung –
eine vergleichende Betrachtung aktueller Entwicklungen.
Diplomarbeit, Otto-von-Guericke-Universität Magdeburg, 1996.

Vorwort

Die vorliegende Arbeit wurde im Frühjahr/Sommer 1996 am Institut für Technische Informationssysteme der Otto-von-Guericke-Universität Magdeburg erstellt. Die Arbeit stellte sich zum Ziel, einen Überblick über aktuelle Entwicklungen im Bereich des föderierten Transaktionsmanagements zu geben. Dazu wurden die drei Hauptprobleme einer föderierten Transaktionsverwaltung untersucht: das globale Serialisierbarkeitsproblem, das globale Atomaritäts- und Fehlertoleranzproblem und das globale Verklemmungsproblem. Lösungen für die einzelnen Problemstellungen und vollständige Lösungen für alle Probleme werden vorgestellt und auf ihre Eignung zum Einsatz in föderierten Datenbanksystemen untersucht. Dabei wird besonderes Augenmerk auf die für föderierte Systeme geforderten Eigenschaften der Heterogenität und Autonomie gelegt und eine vergleichende Bewertung unterschiedlicher Realisierungsansätze im Hinblick auf diese Eigenschaften gegeben.

Danksagung

An dieser Stelle möchte ich allen meinen Dank aussprechen, die mich bei der Erstellung dieser Diplomarbeit tatkräftig unterstützt haben. Mein besonderer Dank gilt Dr. Stefan Conrad und Dipl.-Inform. Can Türker, die diese Arbeit betreuten und mir durch eine Reihe von Anregungen und fachlichen Diskussionen hilfreich zur Seite standen. Außerdem möchte ich Prof. Dr. rer. nat. habil. G. Saake danken, der mir die Bearbeitung dieser Diplomarbeitsthematik innerhalb der Arbeitsgruppe “Datenbanken” ermöglichte.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Abkürzungen

AC	*atomic commitment*, atomares "Commitment"
ACA	*avoid cascading aborts*, Vermeidbarkeit kaskadierender Abbrüche
ACID	*atomicity-consistency-isolation-durability*, Atomarität-Konsistenz-Isolation-Dauerhaftigkeit
AESO	*analogous execution and serialization order*, analoge Ausführungs- und Serialisierungsreihenfolge
CCSR	*chain conflicting serializability*, Konfliktkettenserialisierbarkeit
CC	*concurrency control component*, Sychronisationskomponente
CDBS	*component database system*, Komponentendatenbanksystem
CDBMS	*component database management system*, Komponentendatenbankmanagementsystem
CDM	*canonical data model*, kanonisches Datenmodell
CO	*commitment ordering*, "Commitment"-Sortierung
COCO	*commitment ordering coordinator*
CORCO	*commitment ordering recovery coordinator*
CSR	*conflict serializability*, Konfliktserialisierbarkeit
CTM	*conservative ticket method*, konservative Ticketmethode
C2PL	*conservative two phase locking*, konservatives Zwei-Phasen-Sperren
DBS	*database system*, Datenbanksystem
DBMS	*database management system*, Datenbankmanagementsystem

DM	*data model*, Datenmodell
DML	*data manipulation language*, Datenmanipulationssprache
DS	*data structure*, Datenstruktur
FDBS	*federated database system*, föderiertes Datenbanksystem
FDBMS	*federated database management system*, föderiertes Datenbankmanagementsystem
GSG	*global serialization graph*, globaler Serialisierungsgraph (der OTM)
GTM	*global transaction manger*, Globaler Transaktionsmanager
GWFG	*global wait for graph*, globaler "wait-for"-Graph
HSR	*hybrid serializability*, hybride Serialisierbarkeit
ITM	*implizit ticket method*, Implizite Ticketmethode
LLT	*long lived transaction*, langlebige Transaktion
LWFG	*local wait for graph*, lokaler "wait-for"-Graph
MDBS	*multidatabase system*, Multidatenbanksystem
OTM	*optimistic ticket method*, Optimistische Ticketmethode
PCG	*potential conflict graph*, potentieller Konfliktgraph
PRED	*prefix reducibility*, Präfix-Reduzierbarkeit
RAS	*resource access scheduler*
RC	*recoverability*, Rücksetzbarkeit
RED	*reducibility*, Reduzierbarkeit
RG	*rigorousness*, Rigorosität
RM	*recovery manager*, Recovery-Manager
R2PL	*rigorous two phase locking*, rigoroses Zwei-Phasen-Sperren

SGT	*serialization graph testing*, Serialisierungsgraphentester
SP	*serialization point*, Serialisierungspunkt
SQL	*structured query language*
SR	*serializability*, Serialisierbarkeit
SSR	*shared serializability* Teilungsserialisierbarkeit
S-S2PL	*strong strict two phase locking*, strenges striktes (rigoroses) Zwei-Phasen-Sperren
StRC	*strong recoverability*, strenge Rücksetzbarkeit
StSR	*strong serializability*, strenge Serialisierbarkeit
S2PL	*strict two phase locking*, striktes Zwei-Phasen-Sperren
TO	*timestamp ordering*, Zeitstempelsortierung
TSG	*transaction site graph*, Transaktion-Knoten-Graph
TTS	*transaction termination scheduler*
TV	Transaktionsverwaltung
USG	*undecided serialization graph*, Serialisierungsgraph für nicht beendete Transaktionen (bei COCO und CORCO)
VSR	*view serializability*, Viewserialisierbarkeit
WFCG	*wait for commit graph*, "wait-for-commit"-Graph
WFG	*wait for graph*, "wait-for"-Graph
2PC	*two phase commitment*, Zwei-Phasen-"Commitment"
2PL	*two phase locking*, Zwei-Phasen-Sperren

Kapitel 1

Einleitung

Die Anforderungen an die Verarbeitung großer Datenmengen in Datenbanksystemen haben sich in den letzten Jahren drastisch gewandelt. Die klassische Datenverarbeitung erfolgt traditionellerweise zentralisiert, eine Datenbank wird in einem Rechner gehalten und verwaltet, die Transaktionen der Anwendungsprogramme sind auf die Arbeit mit einem Datenbanksystem beschränkt. Dieser klassische Ansatz verliert jedoch zunehmend an Bedeutung, da sich mit dem Fortschritt der Kommunikations- und Datenbanktechnologien auch die Aufgabenstellungen und Anforderungen der Nutzer eines Datenbanksystems entwickeln. Die gegenwärtige Situation der Datenverarbeitung ist dadurch charakterisiert, daß eine wachsende Anzahl von Anwendungen Zugriffs- und Manipulationsmöglichkeiten für Daten fordert, die in verschiedenen existierenden Datenbanken residieren, wobei die einzelnen Datenbanken wiederum in verschiedenen heterogenen Hard- und Software-Umgebungen gehalten werden und über die Knoten eines Computernetzwerks verteilt vorliegen können.

Die einzelnen Datenbanken sind dabei in der Regel unabhängig voneinander entwickelt worden, d.h. bei der Wahl der Datenbanksysteme, der Organisation der Daten- und Zugriffstrukturen und der Implementation von Anwendungen wurde auf eine bestimmte, zu realisierende Aufgabenstellung fokussiert, eine mögliche spätere Integration in ein System zur Lösung globalerer Aufgabenstellungen wurde in der Regel nicht berücksichtigt. Ein Problem bildet dabei die Heterogenität der einzelnen Umgebungen, Datenbankmanagementsysteme nutzen verschiedene Datenmodelle, Datendefinitionen und Datenmanipulationsmöglichkeiten und operieren in verschiedenen Systemumgebungen. Auch die Heterogenität der Datenbestände stellt ein Problem für die zu realisierenden Anwendungen dar, semantisch identische Daten können in den einzelnen Datenbanken verschiedene physische und logische Repräsentationen haben und sogar unterschiedliche Werte aufweisen.

Eine Möglichkeit zur Überwindung dieser Probleme bildet die *physische Integration* all jener Daten in eine Datenbank, die für die Realisierung einer bestimmten Anwendung erforderlich sind. Jedoch ist diese Lösung in der Regel ungeeignet, da sie nicht nur teuer ist und eine Konvertierung der bestehenden Anwendungen verlangt, sondern auch

unnötige Datenduplikate erzeugt und eine unabhängige Wartung der einzelnen Datenbestände unmöglich macht.

Eine alternative Möglichkeit bietet die *logische Integration* aller benötigten Daten in eine logische Datenbank. Eine solche Integration soll beim Anwender den Eindruck einer einzigen Datenbank erwecken und die Schwierigkeiten der Arbeit mit unterschiedlichen Datenbankmanagementsystemen und Zugriffsmethoden vor dem Nutzer verbergen. Die logische Datenbank stellt einem Anwender bzw. einem Anwendungsprogramm einen einheitlichen Zugriff auf Daten der verschiedenen Datenbanken zur Verfügung, ohne zu verlangen, daß die Daten in eine neue Datenbank migriert werden müssen oder der Nutzer Charakteristika der einzelnen Systeme kennen muß.

Ein architektonischer Ansatz, der zur Lösung der angesprochenen Probleme und zur Realisierung einer logischen Integration vorgeschlagen wurde, sind die *föderierten Datenbanksysteme*. Ein föderiertes Datenbanksystem besteht dabei aus einer Menge unabhängiger, existierender Datenbanksysteme, die einen einheitlichen Zugriff auf die von ihnen verwalteten Daten erlauben.

Bei der Umsetzung von Ansätzen zur logischen Integration von heterogenen, verteilten und autonomen Datenbanksystemen und bei der Arbeit mit diesen Umsetzungen treten eine ganze Reihe von Problemen auf, deren Lösung Ziel aktueller Forschungsarbeiten auf dem Gebiet der Datenbanktechnik ist. Problemstellungen, die in den letzten Jahren bearbeitet wurden und noch bearbeitet werden, sind beispielsweise Aspekten der Integration der Schemata der einzelnen involvierten Datenbanksysteme und der Überwindung der semantischen Heterogenität der Daten gewidmet. Untersuchungen zur Anfrageoptimierung in derartigen integrierten Datenbanken spielen ebenso eine Rolle, wie der Einsatz objektorientierter Datenbanksysteme in logischen Integrationen.

Ein weiteres bedeutendes Problem in derartigen Umgebungen stellt die Schaffung eines *globalen Transaktionsmanagements* dar. Anliegen einer solchen globalen Transaktionsverwaltung ist es, für Operationsfolgen von Anwendungen, die auf verschiedene Datenbanken in einer logischen Integration zugreifen, die aus der klassischen Datenbanktechnik bekannten ACID-Transaktionseigenschaften zu sichern. In [Bre90] wird die Aufgabe des globalen Transaktionsmanagements wie folgt beschrieben:

> *"Sicherung der globalen Konsistenz*[1] *und Verklemmungsfreiheit in einem Multidatenbanksystem*[2] *in der Gegenwart lokaler Transaktionen (d.h. Trans-*

[1] Gemeint ist hierbei die Gewährleistung einer konsistenzerhaltenden Abarbeitung der Operationen einer Applikation, die über die logische Integrationsebene auf die verschiedenen Datenbanken zugreift. Es wird davon ausgegangen, daß jede Transaktion, die auf der Integrationsebene ausgeführt wird, deren Konsistenzbedingungen erhält. Die Transaktionsverwaltung muß nun sicherstellen, daß auch durch eine parallele Ausführung mehrerer solcher Transaktionen aus Sicht der logischen Integration keine Konsistenzverletzungen auftreten.

[2] In [Bre90] wird ein System, das eine logische Integration mit den zuvor genannten Eigenschaften realisiert, "Multidatenbanksystem" genannt. Die Charakterisierung eines Multidatenbanksystems entspricht im wesentlichen der Charakterisierung der von uns untersuchten "föderierten Datenbanksysteme". Die wesentlichen Charakteristika und Beziehungen der einzelnen Ansätze werden im Kapitel 2 noch näher ausgeführt.

aktionen außerhalb der globalen Kontrolle des Multidatenbanksystems) und in Anbetracht der Unfähigkeit der lokalen Datenbanksysteme, die Ausführung der Multidatenbanktransaktionen (sogenannter "globaler Transaktionen") zu koordinieren, unter der Annahme, daß keine Änderungen am Design der lokalen Datenbankmanagementsysteme erlaubt sind."

Die Schwierigkeiten bei der Umsetzung dieser Aufgabenstellung liegen vor allem in der geforderten Autonomie der einzelnen beteiligten Datenbanksysteme. Ebenfalls sehr problematisch ist der Einfluß lokaler Transaktionen, d.h. Transaktionen, die nicht über die logische Integration auf eine Datenbank zugreifen, sondern direkt von einem lokalen Anwender im Datenbanksystem ausgeführt werden, auf die Ausführungen von Transaktionen, die auf der logischen Integrationsebene arbeiten.

In den nachfolgenden Kapiteln wird ausführlich auf die auftretenden Probleme der Transaktionsverwaltung in föderierten Umgebungen eingegangen und es wird eine Reihe von partiellen und vollständigen Lösungen für diese Probleme vorgestellt. Die dabei untersuchten Ansätze gehen mitunter von recht unterschiedlichen Voraussetzungen aus. Dies liegt daran, daß die verschiedenen Publikationen unterschiedliche Entwicklungsrichtungen bei der Realisierung eines solchen Transaktionsmanagements vorschlagen und mit Blick auf die Umsetzbarkeit eine Vielzahl von Restriktionen unterschiedlichster Art vornehmen. Ziel dieser Arbeit ist es, einen Überblick über die vorgeschlagenen Entwicklungsrichtungen und Ansätze zur Schaffung einer föderierten Transaktionsverwaltung zu geben und einzelne Lösungen aus diesem Spektrum von Ansätzen zu präsentieren. Dem Leser soll ein Eindruck von der Komplexität der Aufgabenstellung vermittelt werden, indem zunächst die Problemsituationen beim föderierten Transaktionsmanagement detailliert erläutert werden, um dann zu zeigen, wie mit Hilfe bestimmter Einschränkungen die Probleme teilweise oder vollständig gelöst werden können.

Die Arbeit gliedert sich dazu wie folgt: Eine Einführung in den Bereich der föderierten Datenbanksysteme erfolgt im Kapitel 2. Dazu sollen zunächst erst einmal der Begriff und die Anforderungen eines föderierten Datenbanksystems geklärt werden. Danach wird der föderierte Ansatz mit anderen verteilten und heterogenen Datenbankarchitekturen verglichen und ein Schichtenmodell für die Föderation unterschiedlicher Datenbanksysteme vorgestellt. Kapitel 3 gibt einen kurzen Überblick über die Grundlagen der klassischen Transaktionsverwaltung und stellt ein allgemeines Transaktionsmodell für föderierte Umgebungen vor. Gleichzeitig wird gezeigt, welche Probleme im speziellen von einem föderierten Transaktionsmanagement zu lösen sind. In den Kapiteln 4 bis 7 werden unterschiedliche Problemlösungsrichtungen und -ansätze vorgestellt und Kapitel 8 faßt die vorgestellten Lösungen überblicksartig zusammen. Eine Zusammenfassung der Thematik und ein kurzer Ausblick im Kapitel 9 beschließen diese Arbeit.

Kapitel 2

Föderierte Datenbanksysteme

Ziel dieses Kapitels ist eine Einführung in die Thematik "Föderierter Datenbanksysteme". Dazu sollen zunächst einige Grundbegriffe erläutert und allgemeine Anforderungen an föderierte Datenbanksysteme aufgestellt werden. Anschließend werden die föderierten Datenbanksysteme in den Kontext der verteilten Datenbanksysteme eingeordnet und eine allgemeine Referenzarchitektur [SL90] vorgestellt.

2.1 Anforderungen an föderierte Datenbanksysteme

Allgemein verstehen wir unter dem Begriff *Datenbanksystem* (DBS – "database system") die Kombination eines *Datenbankmanagementsystems* (DBMS – "database management system") und einer oder mehrerer Datenbanken. Das Datenbankmanagementsystem ist dabei die Gesamtheit der Software-Module, die für die Verwaltung einer Datenbank zur Verfügung stehen [HS95]. Ein *föderiertes Datenbanksystem* (FDBS – "federated database system")[1] besteht aus einer Menge kooperierender, aber autonomer *Komponentendatenbanksysteme* (CDBS – "component database system") und einem *föderierten Datenbankmanagementsystem* (FDBMS – "federated database management system"), das für die kontrollierte und koordinierte Manipulation der Komponentendatenbanksysteme verantwortlich ist.

Abbildung 2.1 [HTJ+95] zeigt die Komponenten eines FDBSs. Wesentlich ist, daß die Komponentendatenbanksysteme in mehr als einer Föderation partizipieren können und die einzelnen CDBSe sowohl als zentrales oder verteiltes als auch erneut föderiertes DBMS realisiert sein können. Ein weiterer wichtiger Aspekt föderierter Datenbanksysteme ist die Möglichkeit der Komponentendatenbankmanagementsysteme (CDBMS – "component database management system"), während der Teilnahme an der Föderation eigenständig lokale Applikationen auf einem lokalen Interface zu bearbeiten. Das

[1] Ausgehend vom englischen Fachbegriff "federated" wird in der deutschsprachigen Literatur auch der Begriff "föderative" DBSe oder DBMSe verwendet. Zur korrekten begrifflichen Abgrenzung müßte man von föderierten Datenbanken und föderativen DBSen und DBMSen sprechen, hier soll jedoch durchgängig von föderierten Datenbanken und Systemen die Rede sein.

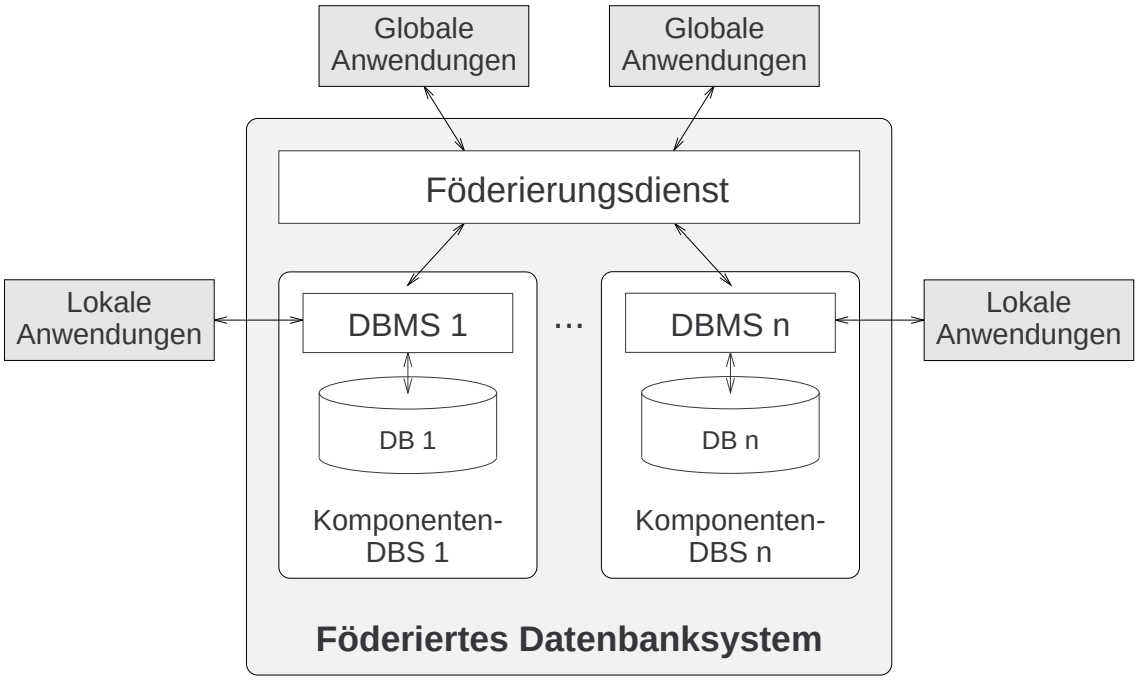

Abbildung 2.1: Komponenten eines FDBSs [HTJ+95]

FDBMS operiert ebenfalls auf der lokalen Schnittstelle der CDBMSe und verhält sich dabei aus Sicht der einzelnen Datenbankmanagementsysteme wie eine lokale Applikation [HTJ+95].

Föderierte Datenbanksysteme bilden eine spezielle Klasse unter den *Multidatenbanksystemen* (MDBS), eine genauere Klassifizierung erfolgt im Abschnitt 2.2. Allgemein zeichnen sich Multidatenbanksysteme durch drei orthogonale Eigenschaften aus: Verteilung, Heterogenität und Autonomie [SL90].

Unter *Verteilung* versteht man die Möglichkeit, daß Daten in einem MDBS in verschiedene Datenbanken aufgeteilt sind, die auf einem oder mehreren Computersystemen gespeichert sein können, die wiederum lokal beieinander oder geografisch getrennt sein können. Für FDBSe ergibt sich die Verteilung aus der Existenz der verschiedenen Datenbanksysteme, die in die Föderation integriert werden sollen.

Heterogenität ergibt sich auf verschiedenen Ebenen, z.B. ist durch unterschiedliche Hardware, System-Software oder Kommunikationsysteme technologische Heterogenität gegeben. Heterogenität in Datenbanksystemen ergibt sich durch unterschiedliche DBMSe und Unterschiede in der Semantik der Daten. Datenbankmanagementsysteme unterscheiden sich vor allem durch die zugrundeliegenden *Datenmodelle* (DM), d.h. die Strukturen, die unterstützten Bedingungen oder Anfragesprachen der einzelnen Datenmodelle, und durch Differenzen auf der Systemebene, d.h. durch unterschiedliche Hard- und Software-Anforderungen, Kommunikationseigenschaften oder Transaktionsverwaltungsmechanismen wie "Concurrency Control", "Recovery" oder "Commit"-Protokolle. Semantische Heterogenität tritt dagegen dann auf, wenn keine Übereinstimmung in der Bedeutung,

der Interpretation oder der beabsichtigten Nutzung gleicher oder verbundener Daten erzielt werden kann.

Die *Autonomie* von verschiedenen Datenbanksystemen ist eine weitere Anforderung an MDBSe. Wesentlich sind dabei drei Arten von Autonomie: Design-, Kommunikations- und Ausführungsautonomie. Designautonomie bezeichnet die Fähigkeit eines Komponentendatenbanksystems, wesentliche Merkmale des Designs selbst festzulegen, u.a. den Diskursbereich, die Repräsentation der Daten, semantische Interpretation und Bedingungen oder die Implementierung, beispielsweise von Transaktionsverwaltungsmechanismen. Vielmehr bezeichnet Designautonomie den Anspruch, daß keine Komponenten der einzelnen DBMSe geändert werden dürfen, um diese CDBSe in die Multidatenbank zu integrieren. Kommunikationsautonomie beschreibt die Eigenschaft eines Komponentendatenbanksystems, eigenständig zu entscheiden, ob und wie mit anderen CDBMSen kommuniziert wird. Ausführungsautonomie ist gegeben, wenn ein CDBMS die vollständige Kontrolle über alle Aktionen auf der zugehörigen Komponentendatenbank hat, d.h. externe Operationen durch das FDBMS werden wie lokale Applikationen behandelt und können dementsprechend auch jederzeit durch das CDBMS zurückgewiesen werden.

In [Hsi92a, Hsi92b] werden eine Reihe historischer, organisatorischer und technologischer Faktoren genannt, die die Entwicklung föderierter Datenhaltungssysteme nötig machten. Aus dieser Vielzahl allgemeiner Faktoren werden fünf generelle Anforderungen an föderierte Datenbanken und -Systeme zusammengefaßt:

- *Transparenter Zugriff auf heterogene Datenbanken in der Föderation*

  Der Nutzer muß die Fähigkeit besitzen, auf die heterogenen Datenbanken wie auf eine homogene Datenbank zuzugreifen, d.h. der Nutzer soll weder die Datenmodelle der einzelnen Systeme noch deren Zugriffssprachen kennen müssen, um auf diese zuzugreifen. Stattdessen arbeitet der Anwender nur in den ihm bekannten Datensprachen und -modellen.

- *Lokale Autonomie der heterogenen Datenbanksysteme*

  Der Eigner einer Datenbank kann diese mit anderen teilen, ohne seine Integritätsbedingungen, Applikationsspezifika und Sicherheitsanforderungen aufzugeben oder einzuschränken. Obwohl vielfach von Nutzern fremder Systeme auf Daten zugegriffen wird und diese manipuliert werden, bleibt die Autonomie des Datenbanksystems erhalten.

- *Multimodell- und multilinguale Fähigkeiten föderierter Datenbanksysteme*

  Die Multimodelleigenschaft bedeutet, daß die Föderation Datenbankmanagementsysteme mit unterschiedlichen Datenmodellen unterstützen soll. Unter der multilingualen Fähigkeit versteht man die Fähigkeit föderierter Datenbanksysteme, in verschiedenen Datenbanksprachen geschriebene Transaktionen verarbeiten zu können.

- *Multibackend-Fähigkeiten*

  Föderierte Systeme müssen in der Lage sein, unterschiedlichste Datenbank-Backends und deren Datenbanken zu unterstützen.

- *Effektive und effiziente Zugriffs- und Synchronisationsmechanismen*

  Effektive und effiziente Zugriffs- und Synchronisationsmechanismen sind Voraussetzung für die Erhaltung der lokalen Autonomie und die Forderung an FDBSe, eine effektive Datenteilung und einen effizienten Ressourcenzusammenschluß zu garantieren.

2.2 Taxonomie

Wie bereits im Abschnitt 2.1 beschrieben, handelt es sich bei föderierten Datenbanksystemen um eine spezielle Klasse von *Multidatenbanksystemen*. In diesem Abschnitt soll nun eine begriffliche Klassifikation der verschiedenen Ansätze entspechend [SL90] gegeben werden.

Man unterscheidet zunächst *zentrale* und *verteilte Datenbanksysteme*. Unter einem zentralen Datenbanksystem verstehen wir ein zentrales Datenbankmanagementsystem, das eine Datenbank auf dem selben Rechner verwaltet. Eine verteilte Datenbank dagegen ist eine Menge verschiedener, logisch verbundener und über ein Rechnernetz verteilter Datenbanken. Ein verteiltes DBMS ist demnach ein Softwaresystem, das das Verwalten einer verteilten Datenbank erlaubt und die Verteilung dem Anwender transparent macht [ÖV91]. Ein *Multidatenbanksystem* unterstützt Operationen auf verschiedenen *Komponentendatenbanksystemen*, die jeweils durch ein CDBMS verwaltet werden und sowohl zentralen wie verteilten Charakters sein können. Ein MDBS wird *homogen* genannt, wenn alle beteiligten Komponentendatenbanksysteme das gleiche CDBMS benutzen, sonst heißt es *heterogen*.

Multidatenbanksysteme sind in *nichtföderierte* und *föderierte* Datenbanksysteme unterteilbar. In nichtföderierten Datenbanksystemen haben die CDBMSe die Eigenschaft, daß sie nicht autonom sind, es existiert nur eine Managementebene und alle Operationen werden einheitlich (uniform) ausgeführt. Sie erscheinen daher dem Anwender oft wie verteilte Datenbanksysteme. Föderierte Datenbanksysteme bestehen aus autonomen Komponentendatenbanksystemen, die an der Föderation teilnehmen, um Anwendern und Applikationen außerhalb des lokalen Systems einen gewissen Zugriff auf die lokalen Daten zu gewähren. Es existiert keine zentrale Kontrolle in einer föderierten Architektur, die einzelnen CDBMSe kontrollieren den Zugriff auf die von ihnen verwalteten Daten eigenständig. Eine föderierte Architektur ist demnach für die Migration einer Menge autonomer und selbständiger Datenbanksysteme zu einem System geeignet, das eine partielle und kontrollierte Datenteilung erlaubt, ohne existierende Applikationen zu beeinflussen. Zusammenfassend bezeichnen föderierte Datenbanksysteme heterogene, verteilte Datenbanksysteme mit autonomen CDBMSen.

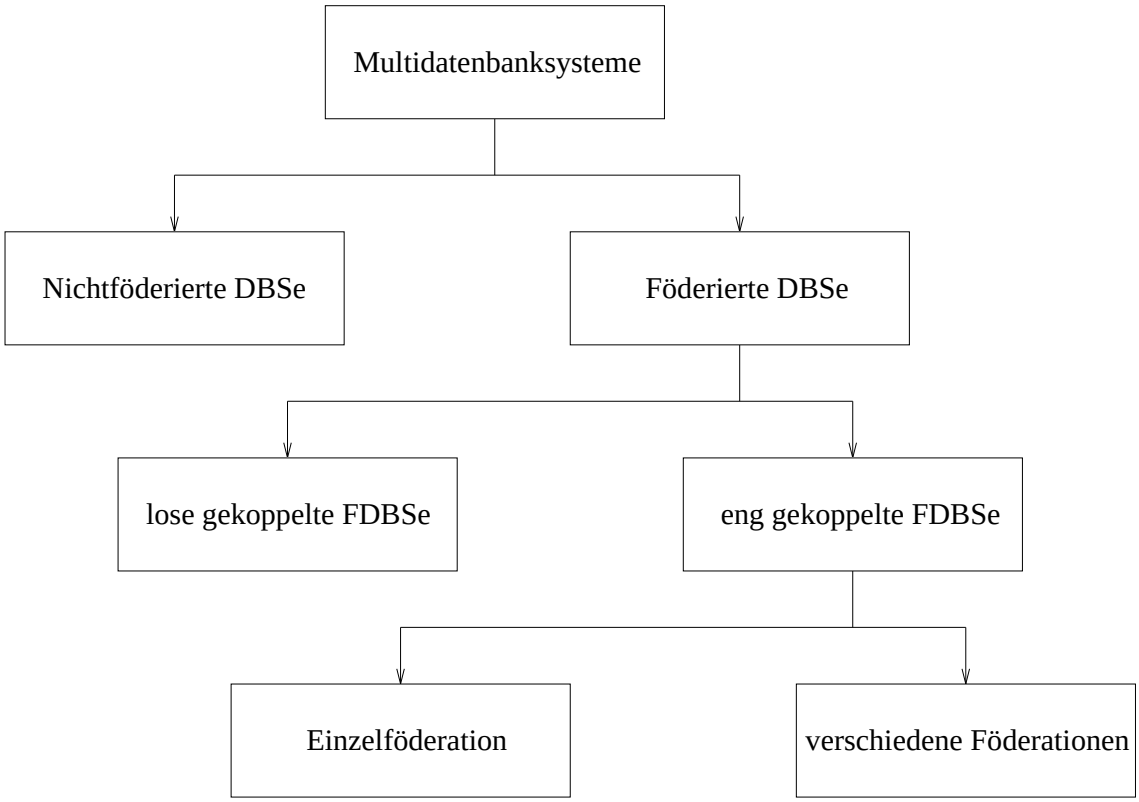

Abbildung 2.2: Taxonomie von MDBSen nach [SL90]

Wir können FDBSe weiter als *lose* oder *eng gekoppelte* Systeme klassifizieren. In lose gekoppelten FDBSen ist der jeweilige Nutzer für das Kreieren und Verwalten einer Föderation verantwortlich. In eng gekoppelten Systemen obliegen diese Aufgaben dagegen einem FDBS-Administrator. Eine Föderation wird durch eine selektive und kontrollierte Integration der verschiedenen Komponenten gebildet und resultiert in der Kreation eines föderierten Schemas. Dementsprechend existieren in einer lose gekoppelten Föderation viele verschiedene föderierte Schemata, wogegen in eng gekoppelten Föderationen nur ein oder mehrere zentral verwaltete föderierte Schemata benutzt werden. Ist nur ein föderiertes Schema in einem eng gekoppelten System erlaubt, handelt es sich um eine *Einzelföderation*. Ein eng gekoppeltes FDBS hat dagegen *verschiedene Föderationen*, wenn das Kreieren und Verwalten mehrerer föderierter Schemata erlaubt ist. Abbildung 2.2 stellt die getroffenen Klassifizierungen bildlich dar.

Mitunter werden in verschiedenen Publikationen zu dieser Klassifikation abweichende Begriffsbestimmungen vorgenommen. So definieren [HM85] eine föderierte Datenbankarchitektur als eine Vereinigung von Komponentendatenbanksystemen in einer lose gekoppelten Föderation zum Teilen und Austauschen von Daten. Sie klassifizieren FDBSe als Systeme mit einer konzeptionell/logisch dezentralisierten Struktur und zentraler oder dezentraler physischer Organisation.

[LMR90] bezeichnen verschiedene autonome Datenbanken, die zusammen ohne ein

globales Schema verwaltet werden, als Multidatenbanken oder *"interoperable databases"*. Systeme, die derartige Datenbanken verwalten, heißen dort Multidatenbanksysteme oder föderierte Datenbanksysteme. Sowohl diese Definition als auch die Definition von [HM85] entspricht der der lose gekoppelten FDBSe in [SL90].

In [Rah94] wird ausgehend von unterschiedlichen Autonomie- und Heterogenitätsanforderungen zwischen *verteilten Transaktionssystemen* und föderierten Datenbanksystemen unterschieden. FDBSe unterteilen sich weiter in lose gekoppelte FDBSe, die hier auch als Multidatenbanksysteme bezeichnet werden, und eng gekoppelte FDBSe. Die Definition der föderierten Datenbanksysteme entspricht der in [SL90] getroffenen Begriffsbestimmung.

[ÖV91] gehen in ihrer Klassifikation von den drei orthogonalen Eigenschaften Heterogenität, Verteilung und Autonomie aus und erreichen durch unterschiedliche Skalierung die verschiedenen DBMS-Klassen. Danach unterscheiden sich föderierte Datenbanksysteme von Multidatenbanksystemen auf der Autonomieebene durch unterschiedlich stark ausgeprägte lokale Autonomie. Föderierte Datenbanksysteme werden in dieser Taxonomie als *semi-autonom* bezeichnet, da die CDBMSe in einer Föderation bereit sind, bei der Bearbeitung von Anfragen über mehrere Datenbanken zu kooperieren, während an MDBSe die Forderung nach vollständiger Autonomie gestellt wird. Ausgehend von unterschiedlichen Autonomieanforderungen läßt sich die Klassifizierung durch Skalieren auf Verteilungs- und Heterogenitätsebene weiter verfeinern. Danach gibt es sowohl homogene und heterogene als auch zentrale und verteilte MDBSe und FDBSe in den entsprechenden Kombinationen.

Als letztes soll hier die Taxonomie aus [BHP92] kurz erwähnt werden. Sie unterscheiden MDBSe von verteilten Datenbanken und interoperierenden Systemen. Multidatenbanken klassifizieren sich weiter in globale Schema-Multidatenbanken, föderierte Datenbanksysteme, Multidatenbank-Sprachsysteme und homogene Multidatenbank-Sprachsysteme. Für die föderierten Datenbanksysteme verwenden [BHP92] die Begriffsbestimmungen und Architekturmerkmale aus [HM85].

2.3 Schemaarchitektur eines föderierten Datenbanksystems

Ein föderiertes Datenbanksystem hat die Aufgabe, eine Menge autonomer und heterogener Komponentendatenbanksysteme miteinander in einer Föderation zu verbinden und so Daten verschiedener Datenbanksysteme globalen Anwendungen zur Verfügung zu stellen [HTJ+95]. In [SL90] wird eine 5-Ebenen-Schema-Architektur vorgestellt, die sich in der Literatur als Referenzarchitektur für die Entwicklung und Bewertung föderierter DBSe durchgesetzt hat.

Grundlage dieser Referenzarchitektur bildet das ANSI/X3/SPARC-Architekturmodell für zentralisierte DBMSe, das eine 3-Ebenen-Schema-Architektur bestehend aus internen, konzeptionellen und externen Schemata vorschlägt [Vos94, HS95]. Ähnlich der 3-

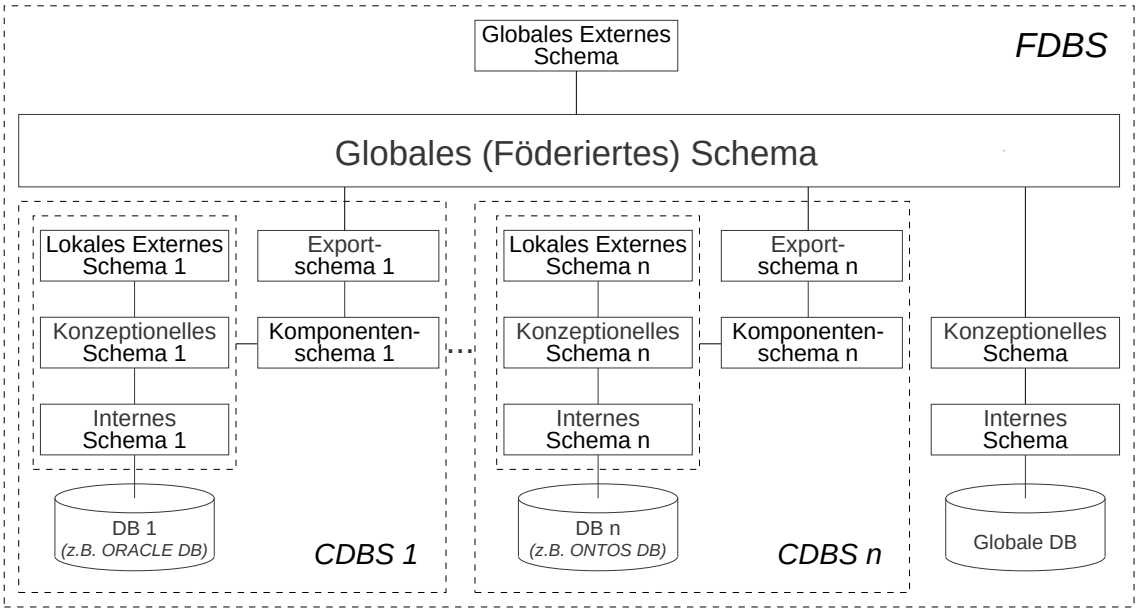

Abbildung 2.3: Schichtenarchitektur eines FDBSs [HTJ+95]

Ebenen-Schema-Beschreibung zentralisierter Datenbanksysteme erfolgt die Beschreibung föderierter Datenbanksysteme in einer fünf Ebenen umfassenden Schemaarchitektur, die aus lokalen, Komponenten-, Export-, föderierten und externen Schemata besteht. Die Abbildung 2.3 aus [HTJ+95] verbindet beide Modelle.

Die konzeptionellen Schemata der Komponentendatenbanken entsprechend der 3-Ebenen-Architektur bilden die lokalen Schemata der Komponentendatenbanken in der 5-Ebenen-Notation. Sie werden im Datenmodell des jeweiligen DBMSs ausgedrückt. Die Komponentenschemata werden durch Translation der lokalen (konzeptionellen) Schemata in ein gemeinsames, kanonisches Datenmodell (CDM – "common/canonical data model") erzeugt. Ein Transformationsprozessor sorgt für die Schematranslation ins CDM und damit für Datenmodelltransparenz. Aus einem Komponentenschema lassen sich entsprechend [SL90] durch Filterprozessoren ein oder mehrere Exportschemata erzeugen. Diese bilden jeweils Untermengen des zugrundeliegenden Komponentenschemas und stellen die Informations- und Operationsmengen dar, die Anwendern der Föderation zur Verfügung gestellt werden sollen. Die verschiedenen Exportschemata lassen sich dann zu einem föderierten Schema integrieren. Die Schemaintegration wird mittels sogenannter Konstruktionsprozessoren durchgeführt und dient der Überwindung der Heterogenität der Schemata. In der Abbildung 2.3 [HTJ+95] wird von einem eng gekoppelten FDBS mit Einzelföderation ausgegangen. Dementsprechend existieren dort nur ein globales (föderiertes) Schema und nur ein Exportschema pro CDBS in der Architektur. Prinzipiell sind jedoch in der 5-Ebenen-Schema-Architektur verschiedene föderierte Schemata möglich, um z.B. lose gekoppelte FDBSe oder eng gekoppelte FDBSe mit verschiedenen Föderationen nach Abschnitt 2.2 darstellen zu können. Für derartige Systeme sind dementsprechend auch Architekturen mit verschiedenen Exportschema-

ta über einem Komponentenschema sinnvoll. Ein globales externes Schema wird für einen Anwender oder eine Applikation bzw. eine Klasse von Anwendern oder Applikationen definiert. Es wird durch einen Filterungsprozeß erzeugt und spezifiziert in der Regel eine Untermenge von Informationen des globalen föderierten Schemas, eventuell verbunden mit zusätzlichen externen Integritätsbedingungen und Zugriffskontrollmechanismen. Während Komponenten-, Export- und föderierte Schemata im kanonischen Datenmodell ausgedrückt werden, können externe Schemata eigenständige Datenmodelle unterstützen, d.h. die Generierung globaler, externer Schemata erzeugt nicht nur Schemaheterogenität sondern mitunter auch Datenmodellheterogenität.

Ähnliche Schemaarchitekturen werden in einer Reihe von Veröffentlichungen vorgeschlagen. [HM85] benutzen eine Referenzarchitektur, in der jedes Datenbanksystem ein Exportschema zur Verfügung stellt. Dafür wird entweder das aktuelle konzeptionelle Schema herangezogen oder aus diesem durch Ausschließen bestimmter privater Daten (privates Schema) abgeleitet. Importschemata vereinigen und gruppieren Daten von verschiedenen Exportschemata und definieren so die Daten, die Anwender der Föderation manipulieren dürfen.

[LMR90] teilen ihre Referenzarchitektur für MDBSe in 3 Ebenen. Die interne Ebene besteht aus den existierenden Datenbanken, die durch die zugehörigen DBMSe verwaltet werden. Ihre Beschreibung erfolgt mittels physischer und interner logischer Schemata. Auf der konzeptionellen Multidatenbankebene beschreiben konzeptionelle Schemata die Datenbanken, die an der Kooperation teilnehmen werden. Externe Datenbankschemata spezifizieren auf der externen Ebene Sichten auf die kooperierenden Datenbanken, die den Anforderungen der jeweiligen Nutzergruppen entsprechen. Die einzelnen Architekturen sind einander ähnlich und lassen sich teilweise aufeinander abbilden, wie [LMR90] beispielhaft zeigen.

Kapitel 3

Transaktionsverwaltung in FDBSen

Innerhalb dieses Kapitels soll eine kurze Einführung in die Thematik "Transaktionsverwaltung" (TV) im allgemeinen und mit besonderer Fokussierung auf föderierte Datenbanksysteme gegeben werden. Zunächst soll dazu der Begriff "Transaktion" definiert und die mit ihm verbundenen ACID-Eigenschaften erläutert werden. Anschließend wird ein allgemeines Transaktionsmodell für FDBSe vorgestellt und die Probleme, die im speziellen bei der föderierten Transaktionsverwaltung zu lösen sind, erläutert.

3.1 Einführung in die Transaktionsverwaltung

Die Transaktionsverwaltung ist eine Thematik, auf die man immer wieder im Bereich der Datenbanken oder Informationssysteme stößt und der im Laufe der Weiterentwicklung dieser Systeme eine immer größere Bedeutung zuteil wurde. Die Transaktionsverwaltung vereinigt Synchronisations- und Fehlertoleranzmechanismen, deren effiziente Implementierung grundlegend für die Leistungsfähigkeit derartiger Systeme ist. Die Synchronisationsmechanismen dienen dem gleichzeitigen Zugriff mehrerer Benutzer oder Applikationen auf einen gemeinsamen Datenbestand ohne gegenseitige Beeinflußung oder gar Störung. Fehlertoleranzmechanismen gewährleisten, daß das System mit bestimmten, im laufenden Betrieb möglicherweise auftretenden Fehlersituationen automatisch, d.h. ohne äußere Hilfe, zurecht kommt, ohne den Betrieb nachhaltig zu beeinflussen oder zu behindern. Daß die Transaktionsverwaltung für Datenbanken und Informationssysteme von existentieller Bedeutung ist, läßt sich auch daran erkennen, daß auch in vielen anderen Bereichen der Informatik, wie z.B. in Betriebssystemen, Transaktionskonzepte zunehmend eingesetzt werden und einen hohen Stellenwert haben [VG93].

In der Datenbanktechnik wird unter einer Transaktion eine Folge von Datenbankoperationen verstanden, die zu Zwecken der korrekten Synchronisation und der Gewährleistung von Fehlertoleranz bestimmten Anforderungen unterworfen ist, die allgemein als *ACID-Eigenschaften* bekannt sind [BHG87, VG93, HS95]:

- **Atomarität (atomicity):**

  Die Operationsfolge wird entweder vollständig oder gar nicht ausgeführt, d.h. Effekte von Datenbankoperationen werden nur dann nach außen sichtbar gemacht, wenn die Abarbeitungsfolge vollständig beendet werden konnte und dabei keine Fehler aufgetreten sind. Mußte dagegen die Operationsfolge nach einem Fehler abgebrochen werden, darf sie keine Zwischenzustände in der Datenbank hinterlassen, d.h. nach außen soll die Datenbank so erscheinen, als wäre die Transaktion nie gestartet worden.

- **Konsistenz (consistency):**

  Transaktionen sind konsistenzerhaltende Operationsfolgen, d.h. wird eine Transaktion auf einem konsistenten Datenbankzustand gestartet, so hinterläßt sie nach erfolgreicher Abarbeitung auch einen konsistenten Zustand, bricht sie dagegen ab, so wird aufgrund der Atomarität der zulässige Ausgangszustand wiederhergestellt. Eine Transaktion muß somit alle geltenden Integritätsbedingungen erfüllen.

- **Isolation (isolation):**

  Transaktionen laufen in einem simulierten Einbenutzerbetrieb ab, d.h. das Ergebnis einer Transaktion muß dem der zugehörigen isolierten Bearbeitung entsprechen, auch wenn nebenläufige Transaktionen auf denselben Datenbestand zugegriffen haben.

- **Dauerhaftigkeit/Persistenz (durability):**

  Die Dauerhaftigkeit oder Persistenz der Ergebnisse einer Transaktion fordert, daß nach einem erfolgreichen Abarbeitungsende die Transaktionsergebnisse dauerhaft in der Datenbank stehen und sie so auch anschließend auftretende Hard- oder Softwarefehler überleben.

Gewährleistet eine Transaktionsverwaltung die oben genannten ACID-Eigenschaften, so hinterlassen fehlerhaft abgebrochene Transaktionen aufgrund der atomaren Abarbeitung keine Spuren in der Datenbank. Persistenz der Ergebnisse sichert, daß im Falle des Wiederanlaufs nach einem Systemfehler der jüngste konsistente Datenbankzustand hergestellt wird. Die Isolationsforderung verhindert "Anomalien" im Mehrbenutzerbetrieb, wobei aus Effizienzgründen keine echte Isolation gefordert wird, sondern nur eine Äquivalenz der realen Abarbeitungsergebnisse zu einer isolierten Ausführung. Lokale zentralisierte Datenbanksysteme sichern die Konsistenz- und Isolationseigenschaft durch die Gewährleistung von (Konflikt-) Serialisierbarkeit, wogegen Atomarität und Dauerhaftigkeit mittels sogenannter "Recovery"-Mechanismen sichergestellt werden.

An dieser Stelle sollen die wesentlichen Grundlagen der Transaktionsverwaltung in Datenbanksystemen für zentrale Systeme kurz erläutert werden. Der zentrale Begriff der Transaktionsverwaltung in Datenbanksystemen ist der *Schedule* (teilweise auch *History*), der eine sequentielle Ausführungsreihenfolge einer Menge von Datenbankoperationen beschreibt. Die zentrale Komponente der Transaktionsverwaltung zur Erzeugung

einer solchen sequentiellen Ausführungsreihenfolge wird *Scheduler* genannt und realisiert, daß die Operationen einer Menge parallel abzuarbeitender Transaktionen derart "gemischt" werden, daß in der sequentiellen Reihenfolge des Schedules die Operationsreihenfolgen der einzelnen Transaktionen erhalten bleiben und der Schedule zusätzlich bestimmte Eigenschaften erhält. Das grundlegende Modell zur Beschreibung von Schedules ist das *"read-write"-Modell*, das davon ausgeht, daß die Datenobjekte einer Datenbank nur durch atomare Lese- ("read"-) und Schreib- ("write"-) Operationen gelesen bzw. modifiziert werden können. Die Leseoperation (Schreiboperation) eines Datenobjektes x einer Transaktion T_i wird durch $r_i(x)(w_i(x))$ dargestellt. Eine Transaktion ist somit eine Menge von Lese- und Schreiboperationen bestimmter Datenobjekte in einer festgelegten Reihenfolge. Ein Scheduler realisiert damit für eine Menge abzuarbeitender Transaktionen eine "Verzahnung" der "read-/write"-Operationen der einzelnen Transaktionen zu einem sequentiellen Schedule derart, daß die festgelegten Reihenfolgen der Operationen in den einzelnen Transaktionen im Schedule erhalten bleiben. Ein Schedule wird als *vollständiger Schedule* bezeichnet, wenn im Schedule nach der letzten Operation einer Transaktion ein c_i (für "$commit(T_i)$") das erfolgreiche Ende charakterisiert bzw. a_i (für "$abort(T_i)$") den Abbruch der Transaktion anzeigt.

Beispiel 3.1 Gegeben seien zwei Transaktionen, die folgendermaßen definiert sind:

$$T_1 : \ r_1(x)w_1(x)$$
$$T_2 : \ r_2(x)r_2(y)w_2(x)$$

Dann ist die folgende Ausführungsfolge ein möglicher vollständiger Schedule:

$$S_1 : \ r_1(x)r_2(x)w_1(x)r_2(y)c_1w_2(x)c_2$$

□

Nachfolgend sollen einige Eigenschaften von Schedules, die für die Transaktionsverwaltung eine wichtige Rolle spielen, kurz erläutert werden. Zunächst heißt ein Schedule *seriell*, wenn für jedes Transaktionspaar T_i und T_j gilt, daß in dem Schedule alle Operationen von T_i vor den Operationen von T_j ausgeführt werden. Das heißt, bei einer seriellen Ausführung werden die zu verarbeitenden Transaktionen in einer beliebigen Reihenfolge nacheinander abgearbeitet. Ein Schedule heißt *serialisierbar*, wenn er *äquivalent* zu einem seriellen Schedule ist, d.h. die Effekte, die die Abarbeitung eines Schedules in der Datenbank hinterläßt, entsprechen denen, die im Falle einer beliebigen seriellen Ausführung in der Datenbank zu sehen wären. Man unterscheidet zwei wesentliche Arten der Serialisierbarkeit in zentralen Datenbanksystemen: *"Viewserialisierbarkeit"* in dem Fall, in dem *"Viewäquivalenz"* vorliegt, und *"Konfliktserialisierbarkeit"*, wenn *"Konfliktäquivalenz"* zu einem seriellen Schedule herrscht.

Die Viewserialisierbarkeit betrachtet die Datenbankzustände, die eine zu verarbeitende Transaktion "sieht". Diese gesehenen Zustände werden mit Hilfe der *"Liest-von-Relation"* ausgedrückt. Man sagt, daß eine Transaktion T_i in einem Schedule S ein Datenobjekt x von einer Transaktion T_j liest, wenn $w_j(x)$ in S die letzte Schreiboperation

einer nicht abgebrochenen Transaktion auf dem Datenobjekt x vor $r_i(x)$ ist. Für einen Schedule S ist dann die Liest-von-Relation $RF(S)$ folgendermaßen definiert:

$$RF(S) = \{(T_i, x, T_j) \mid T_i \; liest \; x \; von \; T_j \; \}$$

Zwei Schedules S_1 und S_2 sind demnach *viewäquivalent*, wenn sie dieselbe Operationsmenge enthalten und $RF(S_1) = RF(S_2)$ gilt.

Definition 3.1 (Viewserialisierbarkeit) Ein Schedule S ist genau dann viewserialisierbar (VSR – "view serializable"), wenn S zu einem seriellen Schedule viewäquivalent ist. □

Die Konfliktserialisierbarkeit betrachtet dagegen die Konflikte zwischen den Transaktionen eines Schedules. Zwei Transaktionen T_i und T_j stehen in einem Schedule S in einem *Konflikt*, wenn sie Operationen auf demselben Datenobjekt x ausführen und mindestens eine Transaktion x schreibt. Anhand der Definition lassen sich drei Konfliktarten zwischen Transaktionen klassifizieren: $wr-$, $rw-$ und ww-Konflikte. Zwei Transaktionen sind in einem "$write/read$"-Konflikt (wr-Konflikt), wenn eine Transaktion ein Datenobjekt x schreibt, das anschließend von einer zweiten Transaktion gelesen wird (die Definition der anderen beiden Konfliktarten verläuft analog). Für einen Schedule S wird dann die Konfliktrelation $C(S)$ über der Operationsmenge des Schedules folgendermaßen definiert:

$$C(S) = \{(o_i, o_j) \mid o_i \in OP(T_i),\; o_j \in OP(T_j),\; o_i \; und \; o_j \; bilden \; einen \; Konflikt \\ und \; o_i \; steht \; in \; S \; zeitlich \; vor \; o_j\}$$

Das heißt, die Operation o_i ist in der Operationsmenge $OP(T_i)$ der Transaktion T_i enthalten und bildet einen Konflikt zur Operation $o_j \in OP(T_j)$, wobei in S die Operation o_i der Operation o_j vorangeht. Bereinigt man die Relation $C(S)$ um alle Konflikte mit abgebrochenen Transaktionen, erhält man die Relation $conf(S)$. Zwei Schedules S_1 und S_2 sind dann *konfliktäquivalent*, wenn sie dieselbe Operationsmenge enthalten und $conf(S_1) = conf(S_2)$ gilt.

Definition 3.2 (Konfliktserialisierbarkeit) Ein Schedule S ist genau dann konfliktserialisierbar (CSR – "conflict serializable"), wenn S zu einem seriellen Schedule konfliktäquivalent ist. □

Der Schedule S_1 im Beispiel 3.1 ist weder view- noch konfliktserialisierbar. Für die Serialisierbarkeitskriterien gilt, daß die Menge der konfliktserialisierbaren Schedules eine echte Untermenge der viewserialisierbaren Schedules ist, d.h. es gilt $CSR \subset VSR$. Das Kriterium der Konfliktserialisierbarkeit stellt dabei das für einen Scheduler bedeutend leichter zu realisierende Kriterium dar, da dieses Kriterium im Gegensatz zur VSR-Eigenschaft zur Laufzeit effektiv zu überwachen ist und es so eine ganze Reihe von

Mechanismen gibt, die die CSR-Eigenschaft der Ausführungsreihenfolge sichern. Einen solchen Mechanismus bildet das Verwalten eines sogenannten *Serialisierbarkeitsgraphen* $SG(S)$ für einen Schedule S. Der $SG(S)$ ist ein gerichteter Graph mit den Transaktionen aus S als Knotenmenge, der eine gerichtete Kante zwischen zwei Knoten enthält, wenn die korrespondierenden Transaktionen in S in einem Konflikt zueinander stehen. Die Richtung der Kante gibt dabei die Ausführungsreihenfolge der Konfliktoperationen an, d.h. $T_i \rightarrow T_j$ ist im $SG(S)$, wenn $(o_i, o_j) \in C(S)$ gilt. Es ist beweisbar, daß ein Schedule konfliktserialisierbar ist, solange sein Serialisierbarkeitgraph azyklisch ist. Eine ausführlichere Beschreibung des Beweises, der aufgeführten Kriterien und eingeführten Begriffe findet sich in der entsprechenden Literatur, z.B. in [BHG87, VG93]. In den nachfolgenden Ausführungen soll Serialisierbarkeit und Äquivalenz im Sinne von Konfliktserialisierbarkeit bzw. Konfliktäquivalenz verstanden werden.

Die Serialisierbarkeit von Transaktionen sichert im wesentlichen die Konsistenz- und Isolationsanforderungen der Transaktionen in einer Abarbeitungsfolge. Zur Durchsetzung der Atomarität und Dauerhaftigkeit sind zusätzliche Eigenschaften der Schedules zu realisieren. Wurden bei den VSR- und CSR-Eigenschaften nur erfolgreich abgearbeitete Transaktionen betrachtet, sollen nun Scheduleeigenschaften kurz erläutert werden, die auch Transaktionsabbrüche betrachten. Das erste Kriterium dabei sichert, daß eine Transaktion T erst freigegeben wird, nachdem alle Transaktionen, von denen T gelesen hat, freigegeben wurden. Dies gewährleistet, daß die Dauerhaftigkeit von T nach der Freigabe nicht durch den Abbruch einer Transaktion verletzt wird, von der T gelesen hat:

Definition 3.3 (Rücksetzbarkeit) Ein Schedule S heißt *rücksetzbar* (RC – "recoverable"), wenn für jedes Transaktionspaar T_i und T_j gilt: liest T_i in S von T_j und wird T_i in S mit c_i beendet, so wird auch T_j in S mit c_j beendet, und zwar bevor T_i beendet wird. □

Intuitiv ist klar, daß ein Schedule rücksetzbar ist, wenn jede Transaktion erst dann mit "commit" endet, wenn alle Transaktionen mit "commit" beendet wurden, von denen gelesen wurde. Die RC-Eigenschaft impliziert aber auch, daß der Abbruch einer Transaktion T den Abbruch aller Transaktionen nach sich zieht, die von T gelesen haben. Ein solcher Vorgang nennt sich *kaskadierender Abbruch* und kann durch die folgende Scheduleeigenschaft verhindert werden:

Definition 3.4 (Vermeidung kaskadierender Abbrüche) Ein Schedule S *vermeidet kaskadierende Abbrüche* (ACA – "avoiding cascading aborts"), wenn für jedes Transaktionspaar T_i und T_j gilt: liest T_i in S von T_j, so wird T_j in S mit c_j beendet, bevor T_i von T_j liest. □

Transaktionen können somit nur von abgeschlossenen Transaktionen lesen, wodurch kaskadierende Abbrüche nicht auftreten können. Problematisch bleibt jedoch die Sicherung der Atomarität einer Transaktion in dem Fall, in dem geschriebene Werte einer

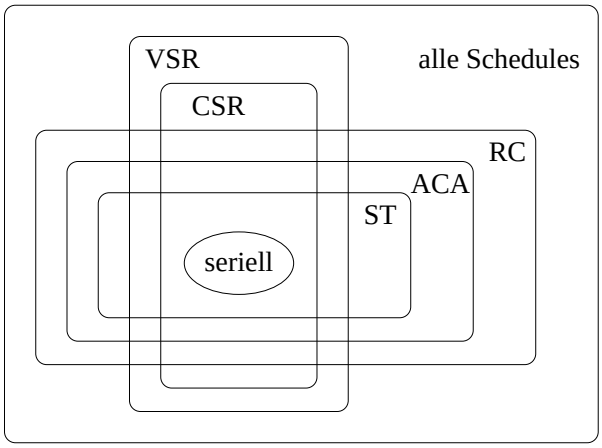

Abbildung 3.1: Beziehungen der vorgestellten Kriterien nach [BHG87]

abgebrochenen Transaktion bereits von anderen Transaktionen überschrieben wurden. Ein ordnungsgemäßes Wiederherstellen des Datenbankzustands vor der Ausführung der abgebrochenen Transaktion ist mit den bisherigen Kriterien nicht möglich. Abhilfe schafft die folgende Scheduleeigenschaft:

Definition 3.5 (Striktheit) Ein Schedule S heißt *strikt* (ST – "strict"), wenn für jedes Transaktionspaar T_i und T_j gilt: T_i darf in S ein zuletzt von T_j geschriebenes Datenobjekt x nur dann lesen oder überschreiben, wenn T_j zuvor mit c_j beendet oder mit a_j abgebrochen wurde. □

In [BHG87] wird bewiesen, daß folgende Inklusionsbeziehung zwischen den Schedulemengen mit den entsprechenden Eigenschaften besteht: $ST \subset ACA \subset RC$. Die Beziehungen zu den Serialisierbarkeitskriterien werden in der Abbildung 3.1 dargestellt.

Zum Abschluß der Erläuterung allgemeiner Grundbegriffe der Transaktionsverwaltung sollen noch einige wesentliche Verfahren, auf die in den nachfolgenden Kapiteln häufig Bezug genommen wird, kurz erläutert werden. Eines der wichtigsten Synchronisationsverfahren zur Realisierung serialisierbarer Schedules ist das *Zwei-Phasen-Sperrprotokoll* (2PL-Protokoll – "two phase locking protocol"). Bei diesem Protokoll muß eine Transaktion vor dem Zugriff auf ein Datenobjekt für dieses Datenobjekt eine Sperre gemäß der durchzuführenden Zugriffsoperation (es werden Lese- und Schreibsperren verwaltet) anfordern. Hat eine Transaktion eine Sperre für ein Objekt erhalten, kann sie die gewünschten Operationen durchführen und nach deren Beendigung die Sperre wieder freigeben. Sperranforderungen anderer Transaktionen werden bis zur Freigabe der Sperre verzögert, wenn sie einen Konflikt verursachen würden. Das 2PL-Protokoll geht nun von einer zweiphasigen Abarbeitung einer Transaktion aus. Zunächst werden während der Sperrphase alle für eine Transaktion benötigten Sperren angefordert und nach deren Erhalt die Datenbankoperationen durchgeführt. In der Entsperrphase werden die erworbenen Sperren suksessiv wieder freigegeben. Keine Transaktion darf erneut eine

Sperre anfordern, nachdem sie bereits eine freigegeben hat. Das 2PL-Protokoll sichert die Serialisierbarkeit von Schedules, wie [BHG87] beweisen, jedoch werden in der Standardversion keine Rücksetzbarkeitskriterien erfüllt. Außerdem sind aufgrund der Charakteristik als sperrendes Verfahren Verklemmungssituationen, sogenannte *"Deadlocks"*, in den Ausführungsreihenfolgen möglich. Dabei warten beispielsweise zwei Transaktionen auf Sperren, die von der jeweils anderen Transaktion gehalten werden. Verklemmungssituationen lassen sich mittels des *konservativen 2PL-Protokolls* (C2PL – "conservative two phase locking") vermeiden, bei dem eine Transaktion alle benötigten Sperren atomar anfordert und erhält. Das *strikte 2PL-Protokolls* (S2PL – "strict two phase locking") sichert dagegen mit der ST-Eigenschaft von Schedules ein wesentliches Rücksetzbarkeitskriterium, indem es alle erhaltenen Sperren atomar zum Terminierungszeitpunkt einer Transaktion freigibt.

Ein weiteres wichtiges Synchronisationsverfahren ist das Zeitstempelverfahren (TO-Verfahren – "timestamp ordering"). In der Grundversion dieses Mechanismus ("Basic TO") wird jeder Transaktion, z.B. bei deren Start, ein Zeitstempel gegeben, der die relative Serialisierungsreihenfolge der Transaktion bestimmt. Folglich werden die Transaktionen entsprechend der Reihenfolge ihrer Zeitstempel serialisiert. Die genaue Arbeitsweise dieses Verfahrens sowie etwaiger Erweiterungen sind der entsprechenden Literatur, z.B. [BHG87, VG93], zu entnehmen.

In verteilten Umgebungen, wie sie z.B. in föderierten Systemen vorliegen, stellt die Sicherung einer atomaren Terminierung verteilt abgearbeiteter Transaktionen ein bedeutendes Problem dar. So muß verhindert werden, daß eine Transaktion, die in verschiedenen Knoten eines verteilten Datenbanksystems abgearbeitet wurde, in einem Knoten mit "commit" endet, wogegen sie in einem anderen Knoten mit "abort" abbricht. Dazu wurden atomare "commit"-Protokolle (AC-Protokolle – "atomic commit protocol") geschaffen, die eine einheitliche Terminierung solcher Transaktionen sichern sollen. Das wohl bekannteste AC-Protokoll ist das *Zwei-Phasen-Commit-Protokoll* (2PC-Protokoll – "two phase commit protocol"), auf das in Kapitel 6 noch ausführlicher eingegangen werden soll.

Die hier gegebene kurze Einführung in Grundlagen der Transaktionsverwaltung soll nur einen groben Überblick über die nachfolgend referenzierten Kriterien und Konzepte geben. Zum vollständigen Verständnis der aufgeführten Ansätze ist sicherlich ein umfassenderes Studium der Eigenschaften und Methoden in [BHG87, VG93] nötig. Prinzipiell wird nachfolgend davon ausgegangen, daß der Leser mit Grundlagen und Grundbegriffen der Transaktionsverwaltung für zentralisierte Datenbanksysteme vertraut ist. So werden in den nachfolgenden Kapiteln und Abschnitten die oben genannten Begriffe und Konzepte als bekannt vorausgesetzt und nur für den Kontext föderierter Systeme neu eingeführt.

Wie bereits im Kapitel 2 geschildert, wird ein föderiertes Datenbanksystem auf einer Menge existierender Komponentendatenbanksysteme aufgesetzt, wobei die lokalen Datenbanksysteme verteilt, heterogen und autonom sein sollen. Besonders die Autonomieeigenschaft spielt für die Transaktionsverwaltung in föderierten Systemen eine besondere

Rolle. Ein föderiertes DBMS betrachtet jedes lokale CDBMS als autonome *"black box"*, wobei, wie bereits angesprochen, Design-, Ausführungs- und Kommunikationsautonomie unterstützt werden soll. Die einzelnen teilnehmenden Komponentendatenbanksysteme können eine unterschiedlich stark ausgeprägte Autonomie aufweisen. Eine Möglichkeit zur Charakterisierung der Autonomieebenen der einzelnen Systeme ist die Definition der Schnittstellen, die die lokalen Datenbanksysteme für Benutzertransaktionen zur Verfügung stellen. So haben beispielsweise Kunden oder einfache Angestellte einer Bank nicht das Recht, auf die Bankdaten mit SQL-Anweisungen zuzugreifen, während bestimmten, entsprechend privilegierten Mitarbeitern dieses Recht eingeräumt werden kann. Die Schnittstellen der lokalen Systeme lassen sich durch die Operationen, die sie in einem föderierten Datenbanksystem unterstützen, kategorisieren. Nachfolgend sollen einige Operationen aufgeführt werden, die lokale CDBS-Schnittstellen in Abhängigkeit von ihren Autonomieebenen unterstützen können. [BGS92] klassifizieren zwei Gruppen von Operationen:

- Transaktionsoperationen:
  - **begin transaction:** Das FDBMS initiiert eine neue lokale Transaktion.
  - **end transaction:** Anforderung zum Beenden einer bestimmten Transaktion.
  - **read / write:** Ausführung der entsprechenden "read-/write"-Operationen (r/w-Operationen) zum Lesen bzw. Schreiben von Datenobjekten.
  - **abort:** Abbruch der entsprechenden Transaktion und Rücksetzen aller Effekte der abgebrochenen Transaktion.
  - **commit:** Erfolgreiches Ende der Transaktion und Festschreiben der durch sie durchgeführten Änderungen.
  - **prepare to commit:** Die identifizierte Transaktion hat ihre Aktionen beendet und ist bereit für ein "commit".
  - **service request:** Anforderung der Ausführung einer Prozedur, z.B. einer festgeschriebenen Folge von Datenbankoperationen einschließlich "begin transaction" und "commit".
- Statusinformationsoperationen:
  - **get-wait-for-graph:** Holen des lokalen *"wait-for"-Graphen* (falls benutzt) für die globale Transaktionsverwaltung.
  - **get-serialization-order:** Holen von Informationen über die lokale Serialisierungsreihenfolge.
  - **inquire:** Abfragen des Status ("committed"/"aborted") einer Transaktion.
  - **disable transaction class:** Bestimmte Transaktionstypen dürfen nicht mit "commit" beendet werden.

Jedes CDBS exportiert somit eine wohldefinierte Menge von *"High-Level"-Operationen*, die durch die Anwendertransaktionen aufgerufen werden können, d.h. das FDBMS arbeitet mittels der zur Verfügung gestellten "logischer" Datenbankoperationen auf der Schnittstellen des jeweiligen CDBS, die Umwandlung in "physische" Seitenzugriffe realisiert dann das CDBMS. Zusätzlich kann ein FDBMS Wissen über interne Fähigkeiten eines partizipierenden Systems nutzen, z.B. ob ein entsprechendes CDBMS die Operationen des 2PC-Protokolls unterstützt oder ob es RC-, ACA- oder ST-Mechanismen für die Transaktionsverwaltung benutzt. Die Möglichkeit des Ausnutzens internen Wissens und der Bereitstellung interner Statusinformationen der CDBMSe in föderierten Umgebungen ist ein entscheidender Grund für die abweichenden Begriffsdefinitionen im Bereich der MDBSe/FDBSe (Abschnitt 2.2), da vielfach die Auffassung besteht, daß derartige Mechanismen der Forderung nach lokaler Autonomie widersprechen.

Die Operationen definieren somit die gesamte Autonomiebreite der teilnehmenden Systeme, angefangen von Systemen, die ausschließlich "Service Requests" bearbeiten, bis zu Systemen, die alle zuvor aufgeführten Operationen individuell an ihrer Schnittstelle zur Verfügung stellen. Folglich ist der einzige Mechanismus, den das FDBMS zur Sicherung bestimmter Eigenschaften der Transaktionsausführung zur Verfügung hat, die Koordination der Operationsreihenfolge der Transaktionen. Generell läßt sich feststellen, daß mit wachsender lokaler Autonomie der CDBSe auch das Problem der Sicherung der globalen Datenbankkonsistenz wächst. Innerhalb dieser Diplomarbeit sollen nur FDBS-Ansätze untersucht werden, deren CDBMSe über *"High-Level"-Autonomie* verfügen, d.h. Systeme, die an ihren lokalen Schnittstellen keine "wait-for"-Informationen oder Serialisierungsreihenfolgen zur Verfügung stellen.

Im folgenden Abschnitt soll nun ein allgemeines Transaktionsmodell für föderierte Datenbanksysteme vorgestellt werden. Anschließend werden einige Probleme der Transaktionsverwaltung mit besonderem Blick auf föderierte Systeme aufgeführt.

3.2 Ein FDBS-Transaktionsmodell

Innerhalb dieses Abschnittes wird ein allgemeines Transaktionsmodell für föderierte Datenbanksysteme vorgestellt, das aus [BGS92] stammt und dort für Multidatenbanksysteme definiert wurde. Die von [BGS92] getroffene Definition für MDBSe entspricht der Begriffsbestimmung für eng gekoppelte FDBSe in unserer Taxonomie (Abschnitt 2.2). Das vorgestellte Transaktionsmodell gilt als das allgemeinste und am häufigsten verwendete Modell im Kontext föderierter Datenbanksysteme und soll für die spätere Betrachtung diverser Transaktionsverwaltungsansätze als Referenzmodell gelten. Beziehen sich nachfolgende Betrachtungen auf alternative Transaktionsmodelle, so wird dies in den entsprechenden Abschnitten explizit erwähnt. In diesem allgemeinen Modell wird angenommen, daß jedes CDBMS an seiner Schnittstelle mindestens die Operationen *"read"* und *"write"* von Datenobjekten und *"commit"* und *"abort"* von Transaktionen unterstützt.

Ein föderiertes Datenbanksystem besteht aus einer Menge existierender, autonomer

CDBMSe, die auf die Knoten $s_1, \ldots, s_n$ mit $(n > 2)$ verteilt sind. Eine Transaktion T_i besteht aus einer Folge von *"read*(r_i)*"*- und *"write*(w_i)*"*-Operationen, die entweder durch ein *"commit*(c_i)*"* oder *"abort*(a_i)*"* abgeschlossen wird. FDBSe unterstützen zwei Arten von Transaktionen:

- **lokale Transaktionen:** Transaktionen, die ausschließlich unter Kontrolle des lokalen CDBMSs ablaufen, d.h. sie werden außerhalb der FDBMS-Kontrolle abgearbeitet und greifen nur auf Daten eines CDBSs zu.
- **globale Transaktionen:** Transaktionen, die unter FDBMS-Kontrolle ablaufen. Eine globale Transaktion besteht aus einer Anzahl globaler Subtransaktionen, die von den CDBSen als gewöhnliche lokale Transaktionen angesehen und verarbeitet werden.

Ein lokaler Schedule S_k im Knoten s_k ist eine Folge von Operationen globaler und lokaler Transaktionen, die deren Ausführungsreihenfolge in dem Knoten s_k widerspiegelt. Eine Transaktion T_i wird in S_k als *"committed"* (*"aborted"*) bezeichnet, wenn S_k die Operation $c_i(a_i)$ enthält, ansonsten ist sie in S_k *aktiv*. Eine Projektion von S_k auf eine Transaktionsmenge $\mathcal{T}$ ist ein Schedule, der nur Operationen von Transaktionen aus $\mathcal{T}$ enthält. Eine *"commit"-abgeschlossene Projektion* ("committed projection") von S_k ist demnach eine Projektion, die nur Operationen von Transaktionen enthält, die in S_k "committed" sind.

Zwei Transaktionen T_i und T_j sind in S_k genau dann in einem *direkten Konflikt*, wenn in S_k die Operation $o_j(x)$ auf die Operation $o_i(x)$ folgt, wobei mindestens eine der beiden Operationen eine "write"-Operation auf dem Datenobjekt x ist, und wenn T_i nicht vor der Ausführung von $o_j(x)$ mit "abort" abgebrochen wird. T_i und T_j sind in S_k genau dann in einem *indirekten Konflikt*, wenn es in S_k eine Transaktionssequenz $T_1, T_2, ..., T_r$ derart gibt, daß T_i in direktem Konflikt zu T_1 steht, T_1 in direktem Konflikt zu T_2 steht, ..., und T_r in direktem Konflikt zu T_j steht. T_i ist mit T_j in einem Konflikt, wenn T_i in einem direkten oder indirekten Konflikt zu T_j steht.

Wir sagen, daß zwei lokale Schedules *äquivalent* sind, wenn sie über derselben Menge lokaler und globaler Transaktionen definiert sind, die gleichen Operationen und die gleiche Menge paarweise, in Konflikt stehender und mit "commit" abgeschlossener Transaktionen haben. Ein Schedule S_k ist *konflikt-serialisierbar*, wenn er äquivalent zu einem seriellen Schedule ist. Ein lokaler *Serialisierbarkeitsgraph* für einen Schedule S_k ist ein gerichteter Graph mit lokalen und globalen Transaktionen, die in S_k "committed" sind, als Knotenmenge und einer Menge von Kanten derart, daß $T_i \rightarrow T_j$ gilt, wenn T_i mit T_j in S_k in Konflikt steht. Ein lokaler Schedule S_k ist genau dann serialisierbar, wenn der lokale Serialisierbarkeitsgraph azyklisch ist [BHG87].

Ein globaler Schedule S ist eine geordnete Menge aller Operationen lokaler und globaler Transaktionen derart, daß für jeden lokalen Knoten s_k die Projektion von S auf die Menge der in s_k ausgeführten lokalen und globalen Transaktionen den lokalen Schedule S_k ergibt. S ist genau dann *global serialisierbar* (GSR – "global serializable"), wenn

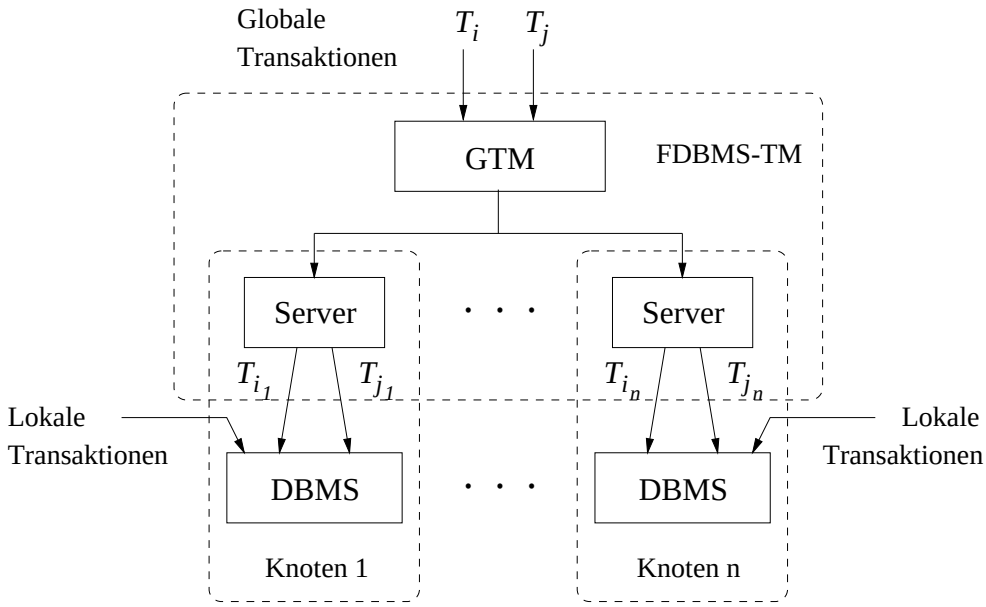

Abbildung 3.2: Transaktionsmodell eines FDBSs nach [BGS92]

es eine totale Ordnung der mit "commit" beendeten globalen Transaktionen gibt, die mit der Serialisierungsreihenfolge der globalen Transaktionen in jedem lokalen DBMS übereinstimmt. Der *globale Serialisierbarkeitsgraph* ist die Vereinigung aller lokaler Serialisierbarkeitsgraphen. Ein globaler Schedule ist genau dann global serialisierbar, wenn der globale Serialisierbarkeitsgraph azyklisch ist [BS88, VG93].

Nach [BGS92] besteht die Transaktionsverwaltungssoftware für ein FDBS, die oberhalb existierender Datenbanksysteme angesiedelt ist, aus einem *globalen Transaktionsmanager* (GTM – "global transaction manager") und einer Menge sogenannter *"Server"*, jeweils einer pro CDBS. Der GTM ist für die Bearbeitung aller globalen Transaktionen verantwortlich und bestimmt, ob und wann die globalen Operationen zu den einzelnen Knoten geschickt werden sollen. In Abhängigkeit von den globalen Daten, die zu bearbeiten sind, bestimmt der GTM, auf welche lokalen Datenbanksysteme zugegriffen werden muß, wobei angenommen wird, daß jedes globale Datenobjekt zu maximal einem lokalen Datenobjekt auf jedem lokalen Knoten korrespondiert. Die globalen Transaktionsoperationen werden durch den GTM über die Server an die lokalen DBMSe weitergereicht, wobei die globalen Transaktionen in globale Subtransaktionen aufgespaltet werden. Hierbei wird davon ausgegangen, daß maximal eine globale Subtransaktion pro CDBS für jede globale Transaktion gebildet wird. Die globalen Subtransaktionen werden von den lokalen CDBMSen als einzelne vollständige Transaktionen verarbeitet, wobei die Ausführung jeder eingereichten globalen Operation dem Server bestätigt wird. In diesem Modell ist es möglich, daß der GTM verschiedene Operationen einer Transaktion parallel ausführt, aber auch eine serielle Ausführung der Operationen einer Transaktion, die die Bestätigung der jeweils vorangegangenen Operation abwartet, ist mit dem Modell vereinbar. Dieses allgemeine Transaktionsmodell wird in der Abbildung 3.2 dargestellt.

Auch die meisten, in verschiedenen Publikationen verwendeten alternativen Transaktionsmanager gehen von dieser zweischichtigen Architektur aus, unterscheiden sich aber in der Umsetzung der einzelnen Komponenten. In [BST90, BST92] wird beispielsweise aus Gründen der Rücksetzbarkeit statt eines allgemeinen Servers pro CDBS ein Server pro lokalem Datenbanksystem und globaler Transaktion eingesetzt. [WV90, VW92] machen dagegen einen alternativen Ansatz zum zentralen GTM, indem sie seine Aufgaben durch dezentrale Koordinatoren ausführen lassen, die jeweils für jede globale Transaktion auf dem entsprechenden Benutzerrechner zu implementieren sind.

Weitere alternative Ansätze sind beispielsweise das *erweiterte Basismodell* und Architekturen mit sogenannten *"Service Interfaces"*. Erstere bieten zusätzliche Operationen (z.B. "prepare to commit") oder nutzen Wissen über die Synchronisationsmechanismen der partizipierenden Systeme. Bei letzteren kommuniziert der GTM über "Servive Requests" mit den lokalen Systemen, d.h. anstatt der r/w-Operationsfolgen sendet der GTM einzelne Dienstanforderungen, die von den lokalen Systemen zur Verfügung gestellt werden und erst in den jeweiligen Komponentendatenbanken erfolgt eine Transformation in "read-/write"-Folgen. Auf derartige Ansätze wurde bereits im Abschnitt 3.1 bei der Definition der unterschiedlichen Autonomieebenen hingewiesen.

3.3 Probleme der Transaktionsverwaltung in föderierten DBSen

Die Hauptaufgabe des GTM besteht in der Sicherung der ACID-Eigenschaften globaler Transaktionen. Probleme bereiten dabei die lokalen Transaktionen, deren Vorhandensein dem GTM nicht bewußt ist und die folglich Probleme für die globale Synchronisation und Fehlertoleranz nach sich ziehen. [BGS92] nennen die drei Hauptprobleme, die im Zusammenhang mit der Transaktionsverwaltung in FDBSen auftreten. Diese Probleme werden in den nachfolgenden Abschnitten kurz skizziert.

3.3.1 Das globale Serialisierbarkeitsproblem

Die verschiedenen CDBMSe verwenden verschiedene Synchronisations- (*"Concurrency Control"*-) Mechanismen, die keine oder unterschiedliche Kontrollinformationen an den Schnittstellen bereitstellen. Existierende Lösungen für homogene verteilte Datenbanken können somit nicht angewendet werden. Da lokale Transaktionen außerhalb der Kontrolle des globalen Transaktionsmanagers ablaufen, kann der GTM einzig durch die Steuerung der Ausführungsreihenfolge globaler Transaktionen globale Serialisierbarkeit garantieren. Jedoch können durch indirekte Konflikte in den verschiedenen teilnehmenden Systemen, die für den GTM nicht erkennbar sind, sogar serielle Ausführungen globaler Transaktionen zu nichtserialisierbaren globalen Schedules führen:

Beispiel 3.2 Ein FDBS bestehe aus zwei Knoten: s_1 mit den Datenobjekten a und b, s_2 mit den Datenobjekten c und d. GT_1 und GT_2 seien zwei globale Transaktionen:

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

Zusätzlich existieren zwei lokale Transaktionen LT_3 und LT_4 auf den Knoten s_1 und s_2:

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

Eine global serielle Ausführung, in der GT_1 auf beiden Knoten mit "commit" endet, bevor GT_2 startet, kann zu folgenden lokalen Schedules führen:

$$S_1 : r_1(a)c_1w_3(a)w_3(b)c_3r_2(b)c_2$$
$$S_2 : w_4(c)r_1(c)c_1r_2(d)c_2w_4(d)c_4$$

Es zeigt sich, daß in Knoten s_1 die Transaktion GT_1 vor GT_2 serialisiert wird ($GT_1 \rightarrow LT_3 \rightarrow GT_2$), wogegen im Knoten s_2 die Serialisierungsreihenfolge umgekehrt ist ($GT_2 \rightarrow LT_4 \rightarrow GT_1$). Folglich ist trotz serieller Ausführung der globalen Transaktionen die globale Serialisierbarkeit verletzt. □

3.3.2 Das globale Atomaritäts- und Fehlertoleranzproblem

Globale Atomarität erfordert, daß entweder alle Subtransaktionen einer globalen Transaktion mit "commit" enden oder alle mit "abort" abgebrochen werden. Atomare "commit"-Protokolle wie das 2PC-Protokoll [BHG87] für verteilte Datenbanken sind nicht anwendbar, da aufgrund der lokalen Autonomie die beteiligten Systeme keinen sichtbaren "prepared to commit"-Zustand zur Verfügung stellen müssen. Da die lokalen DBMSe Ausführungsautonomie besitzen, können sie jederzeit unilateral globale Subtransaktionen abbrechen:

Beispiel 3.3 Ein FDBS bestehe aus zwei Knoten: s_1 mit den Datenobjekten a und b, s_2 mit dem Datenobjekt c. Wir betrachten die folgende globale Transaktion GT_1:

$$GT_1 : r_1(a)w_1(a)w_1(c)$$

Wir nehmen an, daß alle r/w-Operationen von GT_1 erfolgreich abgearbeitet wurden und der GTM nun die "commit"-Anforderung an die beiden Knoten sendet. Knoten s_2 erhält die Anforderung und beendet erfolgreich die Subtransaktion. Knoten s_1 entscheidet sich jedoch dafür, die globale Subtransaktion abzubrechen, bevor die "commit"-Anforderung eintrifft, und setzt die lokalen Effekte von GT_1 zurück. Danach wird die lokale Transaktion LT_2 abgearbeitet:

$$LT_2 : r_2(a)w_2(a)$$

Der resultierende globale Schedule ist inkorrekt, da die Atomarität der globalen Transaktion verletzt ist. Der GTM muß somit versuchen, durch Wiederholung der verlorengegangenen "write"-Operationen die Effekte einer atomaren Ausführung zu sichern. Die lokalen Transaktionsverwaltungsmechanismen betrachten diese wiederholte Ausführung jedoch als neue Transaktion, so daß aus Sicht der lokalen DBMSe die "commit"-abgeschlossene Projektion des Schedules S_1 wie folgt aussieht:

$$CP(S_1) : r_2(a)w_2(a)c_2w_3(a)c_3$$

Aus Sicht des GTM ist jedoch die Operation $w_3(a)$ Teil der globalen Transaktion GT_1, so daß für den GTM die resultierende Ausführung nicht serialisierbar ist ($GT_1 \rightarrow LT_2 \rightarrow GT_1$):

$$r_1(a)r_2(a)w_2(a)c_2w_1(a)c_1$$

□

Ein sichtbarer "prepared"-Zustand bzw. die Bereitstellung einer "prepare to commit"-Operation durch die teilnehmenden DBMSe würde obiges Problem lösen, jedoch gleichzeitig die Ausführungsautonomie der lokalen Systeme verletzen. [BGS92] unterstreichen, daß es eine laufende Debatte über das "Für" und "Wider" eines solchen Zustandes bzw. einer solchen Operation im Zusammenhang mit föderierten Systemen gibt. In dieser Diplomarbeit sollen darum sowohl Ansätze mit Ausnutzung der "prepare to commit"-Operation, als auch Ansätze ohne deren Nutzung vorgestellt werden.

3.3.3 Das globale Verklemmungsproblem

Unter einer *globalen Verklemmung* (einem *globalen Deadlock*) verstehen wir, ähnlich wie in zentralen Datenbanksystemen, eine Situation, in der eine bestimmte Menge lokaler und globaler Transaktionen sich gegenseitig derart blockiert, daß ohne den Abbruch einer oder mehrerer Transaktionen aus dieser Menge keine der Transaktionen zum Abarbeitungsende gelangt. In FDBSen benutzen viele beteiligte Komponenten Sperrmechanismen zur Sicherung der lokalen Serialisierbarkeit. Auch wenn alle CDBMSe in der Lage sind, lokale Verklemmungen zu erkennen und zu behandeln, können globale Verklemmungen auftreten, die durch den GTM nicht erkannt werden können:

Beispiel 3.4 Ein FDBS bestehe aus zwei Knoten: s_1 mit den Datenobjekten a und b, s_2 mit den Datenobjekten c und d. Die lokalen DBMSe beider Knoten benutzen 2PL-Protokolle [BHG87] zur Sicherung der lokalen Serialisierbarkeit. GT_1 und GT_2 seien zwei wie folgt definierte globale Transaktionen:

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

Zusätzlich existieren zwei lokale Transaktionen LT_3 und LT_4 auf den Knoten s_1 und s_2:

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

Folgende globale Verklemmungssituation ist nun denkbar: Die beiden globalen Transaktionen haben jeweils ihre erste "read"-Operation ausgeführt. Danach hat LT_3 auf Knoten s_1 die Operation $w_3(b)$ ausgeführt und wartet nun auf die Freigabe der Sperre für a durch GT_1. Auf Knoten s_2 hat LT_4 die Operation $w_4(d)$ ausgeführt und muß nun ebenfalls auf die Freigabe der Sperre für c warten. Werden jetzt die jeweils zweiten "read"-Operationen der globalen Transaktionen durch den GTM zur Ausführung an die lokalen DBMSe weitergereicht, tritt eine globale Verklemmungssituation ein. Im Knoten s_1 wartet die Transaktion GT_2 auf die Freigabe von Sperren von LT_3, welche wiederum auf Sperren von GT_1 wartet. Im Knoten s_2 dagegen wartet die Transaktion GT_1 auf die Freigabe von Sperren von LT_4, welche wiederum auf Sperren von GT_2 wartet. □

Aufgrund der Designautonomie der beteiligten Systeme werden keine lokalen Kontrollinformationen, wie lokale "wait-for"-Graphen ausgetauscht, so daß der GTM nicht in der Lage ist, globale Verklemmungen zu erkennen.

In den folgenden Kapiteln sollen nun verschiedene Ansätze zur Realisierung einer Transaktionsverwaltung in föderierten Datenbanksystemen vorgestellt werden. Dabei soll untersucht werden, inwieweit die einzelnen Vorschläge in der Lage sind, die hier aufgeführten Probleme zu lösen.

Kapitel 4

Globale Serialisierbarkeit

Innerhalb dieses Kapitels sollen einige Techniken beschrieben werden, die globale Serialisierbarkeit in fehler- und abbruchfreien Umgebungen realisieren. Wir nehmen also an, daß jede globale und lokale Transaktion, die zur Ausführung kommt, erfolgreich abgearbeitet wird. Insbesonders sind unter dieser Annahme keine Abbrüche globaler Transaktionen durch lokale DBMSe, beispielsweise bei lokalen Verklemmungen, erlaubt. Natürlich sind solche Annahmen aus praktischer Sicht nicht realistisch, jedoch sollen sie hier zur Verdeutlichung der Synchronisationsprobleme vorausgesetzt werden. Wie die einzelnen Techniken um Fehlertoleranzmechanismen und Mechanismen zur Verklemmungsbehandlung erweitert werden können, soll in späteren Kapiteln erläutert werden.

Das Schlüsselproblem bei der Sicherung der globalen Serialisierbarkeit stellen die indirekten Konflikte dar, die durch lokale Transaktionen zwischen sonst konfliktfreien globalen Transaktionen entstehen können. Beispiel 3.2 zeigte, daß globale Transaktionen trotz global serieller Ausführung durch indirekte Konflikte in den lokalen DBMSen unterschiedlich serialisiert werden können und so Zyklen im globalen Serialisierbarkeitsgraphen entstehen. Der GTM hat die Aufgabe, geeignete Mittel und Wege zu finden, derartige Zyklen zu vermeiden. Welche Mittel und Wege das im einzelnen sind, hängt vom Wissen des globalen Transaktionsmanagements über die lokalen Transaktionsmechanismen ab. Ausgehend vom Transaktionsmodell aus Abschnitt 3.2 soll zunächst angenommen werden, daß dem GTM nur bekannt ist, daß die partizipierenden DBMSe lokal serialisierbare und verklemmungsfreie Schedules erzeugen. DBMSe, die nur diese Grundvoraussetzung erfüllen, werden auch als *"unlabeled black boxes"* bezeichnet. Realisierungsansätze für globale Transaktionsmanager, die nur von derartigen lokalen DBMSen ausgehen, werden im Abschnitt 4.1 "DBMSe ohne zusätzliche Eigenschaften" vorgestellt. Eine große Anzahl in der Praxis eingesetzter Systeme realisiert jedoch Scheduleeigenschaften, die über die genannten Grundanforderungen hinausgehen. Sind diese zusätzlichen Eigenschaften dem GTM bekannt, so können diese vom globalen Transaktionsmanagement ausgenutzt werden, was zu einer höheren Parallelität und einem besserem Transaktionsdurchsatz führen kann. Realisierungsvorschläge, die von der Ausnutzung zusätzlicher Scheduleeigenschaften ausgehen, werden im Abschnitt 4.2 "DBMSe mit zusätzlichen Eigenschaften" (*"labeled black boxes"*) vorgestellt.

Prinzipiell können die Synchronisationsmechanismen des GTM *optimistischer* oder *pessimistischer* Natur sein. Optimistische Ansätze gehen davon aus, daß bei der Serialisierung keine oder nur wenige Zyklen im globalen Serialisierbarkeitsgraphen auftreten. Treten trotzdem Zyklen auf, so können diese erkannt und durch geeignete Transaktionsabbrüche aufgebrochen werden. Pessimistische Verfahren versuchen Zyklen prinzipiell zu vermeiden, indem Transaktionen, die einen Zyklus verursachen könnten, in der Ausführung verzögert werden.

Zunächst sollen einige Kriterien und Verfahren vorgestellt werden, die von DBMSen als "unlabeled black boxes" ausgehen, d.h. keine Scheduleeigenschaften außer der lokalen Serialisierbarkeit und Verklemmungsfreiheit ausnutzen.

4.1 DBMSe ohne zusätzliche Eigenschaften

Motivierend zeigen [BGS92, VG93], daß aufgrund der begrenzten Kontrollmöglichkeiten des GTM bei der Transaktionabarbeitung sowohl optimistische als auch pessimistische Verfahren ohne zusätzliche Hilfsmittel bei der Lösung des Problems aus Beispiel 3.2 fehlschlagen. Dies verdeutlichen die beiden lokalen Schedules S_1 und S_2, in denen GT_1 jeweils beendet ist, bevor GT_2 gestartet wird. Die Aktionen in eckigen Klammern zeigen die noch zu bearbeitenden Schritte:

$$S_1 : r_1(a)c_1[w_3(a)w_3(b)c_3r_2(b)c_2]$$
$$S_2 : w_4(c)r_1(c)c_1[r_2(d)c_2w_4(d)c_4]$$

Der GTM hat in beiden Knoten die Transaktion GT_1 erfolgreich abgearbeitet. Bei einem pessimistischen Ansatz versucht der GTM, Zyklen im globalen Serialisierbarkeitsgraphen zu vermeiden. Da die globalen Transaktionen nicht in einem direkten Konflikt zueinander stehen, kann ein solcher Zyklus nur durch indirekte Konflikte über lokale Transaktionen entstehen, d.h. der pessimistische GTM muß verhindern, daß durch mögliche indirekte Konflikte unterschiedliche Serialisierungsreihenfolgen in den Knoten entstehen. Da der GTM nur die Kontrolle über die globale Ausführungsreihenfolge der globalen Operationen hat, versucht er, eine zur globalen Ausführungsreihenfolge analoge Serialisierungsreihenfolge zu forcieren. Der GTM verzögert dazu GT_2 so lange, bis alle zu GT_1 überlappend abgearbeiteten Transaktionen ebenfalls beendet wurden, d.h. der GTM verzögert GT_2 bis im Knoten s_2 die lokale Transaktion LT_4 beendet wurde. Jedoch hat der GTM keine Kontrolle über die Abarbeitung von LT_4, so daß er deren Ende nicht erkennt und so der pessimistische Ansatz nicht funktioniert. Der optimistische Ansatz funktioniert ebenfalls nicht, da bei ihm die obigen Schedules vollständig abgearbeitet würden, obwohl GT_2 aufgrund des entstehenden Zyklus abgebrochen werden müßte. Da dem GTM jedoch die lokalen Transaktionen und daraus resultierenden indirekten Konflikte nicht bekannt sind, erkennt er weder die unterschiedlichen lokalen Serialisierungsreihenfolgen noch die Zyklen im globalen Serialisierbarkeitsgraphen. Eine praktische Lösung für dieses Problem ist die Forcierung direkter Konflikte oder anderer Eigenschaften zwischen den

globalen Transaktionen, die unterschiedliche lokale Serialisierungsreihenfolgen unmöglich machen. Nachfolgend werden einige solcher Eigenschaften vorgeschlagen und Methoden vorgestellt, die diese Eigenschaften forcieren.

4.1.1 Konfliktkettenserialisierbarkeit

Wie bereits einleitend erläutert wurde, besteht das Hauptproblem bei der Sicherung der globalen Serialisierbarkeit darin, daß aufgrund indirekter Konflikte zwischen globalen Transaktionen deren lokale Serialisierungsreihenfolgen, die durch die lokalen DBMS-Komponenten festgelegt werden, inkonsistent zur globalen Operationsausführungsreihenfolge sein kann. Die Aufgabe der Transaktionsverwaltung in föderierten Systemen mit CDBMSen ohne zusätzliche Eigenschaften besteht somit darin, Kriterien für das globale Transaktionsmanagement zu schaffen und durchzusetzen, die solche inkonsistenten Serialisierungsreihenfolgen verhindern und einheitliche Serialisierungsreihenfolgen in allen beteiligten CDBMSen sichern. Da der GTM lediglich die Kontrolle über die Ausführungsreihenfolge der globalen Transaktionsoperationen ausüben kann, ist es nötig, globale Transaktionseigenschaften zu forcieren, die unterschiedliche Serialisierungsreihenfolgen in den verschiedenen lokalen DBMSen unmöglich machen. Nachfolgend sollen einige mögliche Kriterien untersucht und Methoden zu deren Durchsetzung vorgestellt werden.

In [ZE93] wird eine theoretische Grundlage für das globale Transaktionsmanagement zur Sicherung globaler Serialisierbarkeit in föderierten Umgebungen gegeben, von deren Komponenten lediglich bekannt ist, daß sie serialisierbare Ausführungsreihenfolgen realisieren. [ZE93] stellen drei Korrektheitskriterien für verteilte, heterogene und autonome Datenbanksysteme vor, die für sich den Anspruch erheben, globale Serialisierbarkeit von Transaktionen sicherzustellen, ohne zusätzliche Restriktionen an die lokalen Ausführungen vorzunehmen und dadurch die lokale Autonomie der beteiligten Komponenten zu verletzen. Diese drei Kriterien sind die Konfliktkettenserialisierbarkeit (CCSR – "chain conflicting serializability"), Teilungsserialisierbarkeit (SSR – "shared serializability") und hybride Serialisierbarkeit (HSR – "hybrid serializability") und werden in diesem Abschnitt vorgestellt.

Zur Erklärung der Kriterien aus [ZE93] wird das Transaktionsmodell des Abschnitts 3.2 benutzt, jedoch sollen hier einige Begriffs- und Notationsergänzungen vorgenommen werden, die die nachfolgenden Ausführungen anschaulicher machen. Wie bereits erläutert, verbindet ein FDBS eine Menge existierender autonomer Komponentendatenbanksysteme ($s_i, 1 \leq i \leq n$), die selbst jeweils eine bestimmte Menge von Datenobjekten D_i verwalten. Zwei Datenbankoperationen werden als *teilend* bezeichnet, wenn sie auf das gleiche Datenobjekt zugreifen. Demnach bezeichnet ein Konflikt zwei teilende Operationen, von denen mindestens eine Operation das Datenobjekt schreibt. D_T bezeichnet die Menge der Datenobjekte, auf die eine Transaktion T zugreift, wogegen OP_T deren Operationsmenge bezeichnet. Ein direkter Konflikt zwischen zwei Transaktionen wird durch $T_i \overset{c}{\sim} T_j$ dargestellt, wobei eine Operation $o_i \in OP_{T_i}$ mit einer Operation $o_j \in OP_{T_j}$ in Konflikt steht. Zwei Transaktionen sind teilend, wenn $D_{T_i} \subseteq (\supseteq) D_{T_j}$ gilt, was man durch

$T_i \subseteq (\supseteq) T_j$ darstellt. Abschließend soll $o_i \prec_S o_j$ zum Ausdruck bringen, daß im Schedule S die Operation o_i vor der Operation o_j ausgeführt wird, wogegen $T_i \prec_S T_j$ bedeutet, daß in S *alle* Operationen von T_i vor den Operationen von T_j ablaufen.

Ziel der drei vorgeschlagenen Kriterien ist die Gewährleistung globaler Serialisierbarkeit ohne den CDBMSen zusätzliche Beschränkungen außer der Sicherung lokaler Serialisierbarkeit aufzuzwingen. Die erschwerenden Effekte der indirekten Konflikte sind bereits im Abschnitt 3.3.1 erläutert wurden.

Die *Konfliktkettenserialisierbarkeit* (CCSR) ist das erste in [ZE93] vorgeschlagene Korrektheitskriterium für globale Subtransaktionen, das in jedem beteiligten Knoten die Ausführungsreihenfolge der Konfliktoperationen globaler Subtransaktionen identisch zur globalen Serialisierungreihenfolge erhält. Dieses Kriterium liefert somit dem GTM ausreichende Bedingungen zur Synchronisation der relativen Serialisierungsreihenfolgen der Subtransaktionen aller globalen Transaktionen. Zur Verdeutlichung des Prinzips der CCSR sollen zunächst Konfliktkettentransaktionen und -serialisierbarkeit definiert werden:

Definition 4.1 (Konfliktkettentransaktionen) Eine Menge lokaler Transaktionen $\mathcal{T}$ bildet eine Konfliktkette, wenn es eine totale Reihenfolge $T_{i_1}, T_{i_2}, \ldots, T_{i_m}$ unter den Transaktionen aus $\mathcal{T}$ derart gibt, daß $T_{i_1} \overset{c}{\sim} T_{i_2} \overset{c}{\sim} \ldots \overset{c}{\sim} T_{i_m}$ gilt. Eine Menge globaler Transaktionen $\mathcal{G}$ bildet eine Konfliktkette, wenn es eine totale Reihenfolge O in $\mathcal{G}$ derart gibt, daß in jedem Knoten s_i $(1 \leq i \leq n)$ die Menge der dort ausgeführten globalen Subtransaktionen eine Konfliktkette bildet, deren Reihenfolge konsistent zu O ist, d.h. in jedem Knoten stehen die dort ausgeführten globalen Subtransaktionen derart miteinander in direkten Konflikten, daß die lokalen Konfliktreihenfolgen verträglich zur Reihenfolge O sind. □

Definition 4.2 (Konfliktkettenserialisierbarkeit) Ein Schedule S heißt konfliktkettenserialisierbar, wenn die Menge der mit "commit" beendeten Transaktionen in S eine Konfliktkette der Reihenfolge O bildet und der Schedule S in der Reihenfolge O serialisierbar ist. □

Die Konfliktkettenserialisierbarkeit ist eine Verschärfung des allgemeinen Serialisierbarkeitsbegriffs, d.h. jeder konfliktkettenserialisierbare Schedule ist auch serialisierbar, die Umkehrung gilt nicht. In Multidatenbankumgebungen kann die Konfliktkettenserialisierbarkeit als Korrektheitskriterium genutzt werden, was mit folgendem Satz zum Ausdruck gebracht werden soll:

Satz 4.1 Es sei S ein globaler Schedule und $\mathcal{G}$ die Menge der globalen Transaktionen in S. Weiterhin sei $S_\mathcal{G}$ die Projektion von S auf die Menge $\mathcal{G}$. Wenn $S_\mathcal{G}$ konfliktkettenserialisierbar ist, dann impliziert die lokale Serialisierbarkeit der Schedules S_k $(1 \leq k \leq n)$ die globale Serialisierbarkeit von S. □

Dieser Satz wird von [ZE93] bewiesen und außerdem eine Graphentestmethode für die CCSR vorgestellt, in der die Zyklenfreiheit des Graphen eines globalen Schedules dessen CCSR-Eigenschaft anzeigt. Auf die Vorstellung des CCSR-Graphen soll hier jedoch aus Gründen der Beschränkung des Umfangs dieser Arbeit verzichtet werden. Statt dessen soll die Idee der CCSR anhand des folgenden Beispiels verdeutlicht werden:

Beispiel 4.1 Ein FDBS bestehe aus zwei Knoten: s_1 mit dem Datenobjekt a, s_2 mit den Datenobjekten b und c. GT_1, GT_2 und GT_3 seien drei globale Transaktionen:

$$GT_1 : w_1(a)r_1(b)$$
$$GT_2 : r_2(a)w_2(c)w_2(b)$$
$$GT_3 : w_3(a)r_3(c)$$

Die globalen Transaktionen bilden eine Konfliktkette in der Reihenfolge $GT_1 \rightarrow GT_2 \rightarrow GT_3$ (ebenso existiert eine Konfliktkette in der Reihenfolge $GT_3 \rightarrow GT_2 \rightarrow GT_1$, weitere mögliche Reihenfolgen gibt es nicht[1]). Zusätzlich existiert eine lokale Transaktion LT_4 auf dem Knoten s_2:

$$LT_4 : w_4(b)w_4(c)$$

Der globale Schedule S bestehe aus folgenden lokalen Schedules S_1, S_2:

$$S_1 : w_1(a)r_2(a)w_3(a)$$
$$S_2 : w_4(b)r_1(b)w_4(c)w_2(c)r_3(c)w_2(b)$$

Offensichtlich ist die Projektion von S auf die globalen Transaktionen $S_{\mathcal{G}}$ konfliktkettenserialisierbar in der Reihenfolge $GT_1 \rightarrow GT_2 \rightarrow GT_3$, da in jedem Knoten die globalen Subtransaktionen Konfliktketten in der gleichen lokale Reihenfolgen bilden, woraus sich nach Satz 4.1 die globale Serialisierbarkeit von S ergibt. □

Wesentlich ist, daß solange die Ausführungsreihenfolge der Konfliktoperationen globaler Transaktionen in allen Knoten identisch gehalten wird, solange kann die globale Serialisierbarkeit durch die lokalen Scheduler nicht verletzt werden. Der Schwierigkeitsgrad bei der Durchsetzung der Kettenkonfliktserialisierbarkeit zwischen globalen Transaktionen hängt von den Schnittstellen zwischen dem GTM und den einzelnen CDBMSen ab. In verschiedenen Publikationen (z.B. [BGS92, MRB+92b]) wird davon ausgegangen, daß globale Transaktionsoperationen (vergleiche Abschnitt 3.1) verfügbar sind, die vom GTM an die lokalen DBMSe weitergereicht werden und deren vollständige Ausführung dem GTM von den CDBSen bestätigt wird. In derartigen Umgebungen sind Methoden zur Sicherung von CCSR-Schedules implementierbar.

[1] Im Knoten s_1 bildet offensichtlich jede beliebige Reihenfolge der drei Subtransaktionen eine Konfliktkette, im Knoten s_2 jedoch schreibt lediglich die Subtransaktion von GT_2, wogegen die Subtransaktionen von GT_1 und GT_3 nur die von GT_2 geschriebenen Werte lesen. Die Bildung einer Konfliktkette ist damit nur dann möglich, wenn die schreibende Subtransaktion die zentrale Position in der Kette einnimmt.

Eine solche Methode zur Forcierung der CCSR-Eigenschaft ist die *Zusatzoperationsmethode*. Wenn in einem lokalen Knoten s_k zwei globale Subtransaktionen T_i und T_j nicht in einem Konflikt zueinander stehen und wenn T_i vor T_j ausgeführt wird und auf ein Datenobjekt x zugreift, so werden an T_j die Operationen $r_j(x)w_j(x)$ angehängt. Durch diesen Zusatz wird ein Kettenkonflikt zwischen T_i und T_j' forciert, ohne die Effekte der Abarbeitung in der Datenbank zu beeinflussen.

Einen alternativen und wesentlich bekannteren Ansatz zur Forcierung von globalen Subtransaktionen mit der CCSR-Eigenschaft bilden die Ticketmethoden aus [GRS91, GRS94].

Die Ticketmethoden von [GRS91, GRS94]

Wie bereits bei der Erläuterung der Konfliktkettenserialisierbarkeit erwähnt, ist die Forcierung direkter Konflikte zwischen den globalen Subtransaktionen eines Knotens ein probates Mittel zur Realisierung einheitlicher lokaler Serialisierungsreihenfolgen. Wird beispielsweise GT_2 im Beispiel 3.2 gezwungen, in jedem Knoten ein Datenobjekt zu lesen, das GT_1 zuvor geschrieben hat, so wird durch den zusätzlichen Konflikt aufgrund der Forderung nach lokaler Serialisierbarkeit GT_1 in jedem Knoten vor GT_2 serialisiert ($GT_1 \rightarrow GT_2$). Diese Eigenschaft nutzen auch [GRS91, GRS94] und verallgemeinern diesen Ansatz in ihren *Ticketmethoden*.

Ausgehend von dem allgemeinen Transaktionsmodell aus Abschnitt 3.2 werden in [GRS91, GRS94] folgende Einschränkungen vorgenommen:

- Für jede globale Transaktion existiert maximal eine Subtransaktion pro CDBS.
- Eine Subtransaktion kann eine "prepare-to-commit"-Operation vor dem "commit" ausführen, wenn die lokale Schnittstelle diese Operation zur Verfügung stellt.
- Subtransaktionen haben einen sichtbaren "prepared-to-commit"-Zustand.

Eine Subtransaktion tritt in den "prepared-to-commit"-Zustand, wenn alle ihre Datenbankoperationen abgearbeitet wurden und verläßt diesen Zustand, wenn eine "commit"-/"abort"-Operation durchgeführt wurde. Der "prepared-to-commit"-Zustand heißt "sichtbar", wenn der GTM entscheiden kann, ob die Transaktion abbrechen oder erfolgreich enden soll. Da nicht alle CDBMSe notwendigerweise einen sichtbaren "prepared-to-commit"-Zustand zur Verfügung stellen, kann er durch geeignete Methoden, z.B. durch "Handshake"-Signale, simuliert werden. [GRS91, GRS94] unterstreichen, daß ein simulierter "prepared-to-commit"-Zustand, obwohl er nicht die vom System garantierten Eigenschaften eines "echten prepared"-Zustand besitzt, in den meisten Fällen völlig gleichwertig zu einem "echten prepared-to-commit"-Status ist. Zur Sicherung der Atomarität der "commit"- bzw. "abort"-Operation nutzt der GTM das 2PC-Protokoll [BHG87], in dem die lokalen Server, hier "Agenten" genannt, die einzelnen Operationen zu den lokalen DBMSen schicken und den Austausch und die Synchronisation der Nachrichten von und zum GTM handhaben.

Von den lokalen DBMSen wird gefordert, daß sie die ACID-Eigenschaften im allgemeinen und lokal serialisierbare und rücksetzbare Schedules im speziellen garantieren und die Server über unilaterale Aktionen informieren.

Die aufgeführten Einschränkungen dienen hauptsächlich den globalen Fehlertoleranzmechanismen, auf die im Kapitel 6 noch näher eingegangen werden soll, und sind folglich für die Untersuchung der globalen Serialisierbarkeit in fehler- und abbruchfreien Umgebungen nicht relevant, sollen hier aber aus Gründen der Vollständigkeit des zugrundeliegenden Modells aufgeführt werden. Für das allgemeine Verständnis der Ticketmethoden ist es vielmehr völlig ausreichend, nur von lokal serialisierbaren und verklemmungsfreien Schedules auszugehen.

a) Die Optimistische Ticketmethode (OTM)

Die erste vorgestellte Methode zur Sicherung globaler Serialisierbarkeit ist die *optimistische Ticketmethode* (OTM – "optimistic ticket method"). Die OTM benutzt logische Zeitstempel (*"Tickets"*) zur Bestimmung der relativen Serialisierungsreihenfolge von Subtransaktionen globaler Transaktionen auf den einzelnen FDBS-Knoten. Dazu wird in jedem Knoten ein solcher Zeitstempel als reguläres Datenobjekt angelegt und jede globale Subtransaktion, die auf einem Knoten arbeitet, gezwungen, auf dem jeweiligen Knoten eine "take-a-ticket"-Operation durchzuführen, die den alten Wert des Zeitstempels liest ($r(ticket)$) und ihn dann inkrementiert zurückschreibt ($w(ticket+1)$). Die Effekte der "take-a-ticket"-Operation lassen sich am folgenden Beispiel erklären:

Beispiel 4.2 S_1 und S_2 seien die beiden Schedules aus dem Beispiel 3.2, die um die "take-a-ticket"-Operation erweitert wurden, wobei t_1 und t_2 die jeweiligen Ticketwerte der Knoten sind:

$$S_1 : r_1(t_1)w_1(t_1)r_1(a)c_1w_3(a)w_3(b)c_3r_2(t_1)w_2(t_1)r_2(b)c_2$$
$$S_2 : w_4(c)r_1(t_2)w_1(t_2)r_1(c)c_1r_2(t_2)w_2(t_2)r_2(d)c_2w_4(d)c_4$$

Schedule S_1 ist weiterhin erlaubt, da er lokal serialisierbar ist. Es ist lediglich ein direkter Konflikt zwischen GT_1 und GT_2 im Serialisierbarkeitsgraphen ($GT_1 \rightarrow GT_2$) hinzugekommen, der mit dem bisherigen Graphen ($GT_1 \rightarrow LT_3 \rightarrow GT_2$) vereinbar ist, d.h. keinen Zyklus bildet. In Schedule S_2 führt jedoch der zusätzliche Ticketkonflikt ($GT_1 \rightarrow GT_2$) zu einem Zyklus im Serialisierbarkeitsgraphen ($LT_4 \rightarrow GT_1 \rightarrow GT_2 \rightarrow LT_4$) und damit zu einem nichtserialisierbaren lokalen Schedule, d.h. die lokale Transaktionsverwaltung müßte zumindest eine Transaktion blockieren oder abbrechen. □

Eine Transaktion kann die "take-a-ticket"-Operation innerhalb ihrer gesamten Laufzeit ausführen, ohne die Korrektheit des Ansatzes zu gefährden. [GRS91, GRS94] beweisen, daß der Wert der Tickets, den die jeweiligen Subtransaktionen lesen, deren Serialisierungsreihenfolge in dem entsprechenden Knoten bestimmen. Zur Sicherung einer

konsistenzerhaltenden globalen Ausführung muß die OTM gewährleisten, daß die Subtransaktionen aller globalen Transaktionen in den beteiligten Datenbanken die gleiche lokale Serialisierungsreihenfolge haben, d.h. es muß sichergestellt werden, daß die Werte der Tickets der Subtransaktionen in den einzelnen Datenbanksystemen die gleiche relative Reihenfolge haben. Da es sich bei der OTM um einen optimistischen Ansatz handelt, werden zunächst alle gestarteten Subtransaktionen abgearbeitet und erst zum Zeitpunkt der "commit"-Operation eine Validierung der Ticketreihenfolgen vorgenommen. Zu diesem Zweck verwaltet der GTM einen *globalen Serialisierungsgraphen* (GSG – "global serialization graph"), der als Knoten die mit "commit" beendeten globalen Transaktionen enthält. Der GSG enthält eine gerichtete Kante zwischen zwei globalen, mit "commit" beendeten Transaktionen $G_i^c \rightarrow G_j^c$, wenn mindestens eine Subtransaktion von G_i^c in einem gemeinsamen Knoten einen kleineren Ticketwert liefert als die dortige Subtransaktion von G_j^c.

Die Abarbeitung und Validierung einer globalen Transaktion G läuft wie folgt: Der GTM setzt zunächst ein Timeout-Intervall für G und reicht die Subtransaktionen zur lokalen Abarbeitung an die CDBMSe weiter. Diese bearbeiten die Subtransaktionen selbständig, bis alle Subtransaktionen den sichtbaren "prepared-to-commit"-Zustand erreicht haben und die Validierung beginnt. Dazu wird in dem anfangs zyklenfreien GSG ein Knoten für G erzeugt und dann die Ticketwerte von G mit denen der bereits beendeten Transaktionen im GSG verglichen. Hat dabei eine Subtransaktion von G in einem Knoten einen kleineren (größeren) Ticketwert als die entsprechende Subtransaktion einer bereits beendeten globalen Transaktion G_i^c, so wird die Kante $G \rightarrow G_i^c$ ($G_i^c \rightarrow G$) in den GSG eingefügt. Bleibt der GSG auch nach dem Vergleich mit allen beendeten Transaktionen zyklenfrei, so ist G validiert und kann mit "commit" beendet werden. Erzeugt die Validierung Zyklen im GSG, so wird G abgebrochen und neu gestartet. Gleiches passiert, wenn Subtransaktionen unilateral abgebrochen wurden oder das Timeout-Intervall abgelaufen ist. Die letzten beiden Fälle sind für die Betrachtung von Atomarität und Dauerhaftigkeit (Kapitel 6) und globalen Verklemmungen (Kapitel 7) relevant und sollen hier nur erwähnt werden. Traten Zyklen im GSG auf, so hat G mit mindestens einer bereits beendeten globalen Transaktion unterschiedliche lokale Ticketreihenfolgen, und die globale Serialisierbarkeit ist verletzt.

Die Wahl des richtigen Zeitpunktes der Ticketoperation während der Subtransaktionslaufzeit hat einen wesentlichen Einfluß auf die Parallelität und die Effizienz der Transaktionabarbeitung in den lokalen DBMSen. So ist beispielsweise in lokalen 2PL-Schedulern als Zeitpunkt für die Ticketoperation das Ende der Transaktionsabarbeitung unmittelbar vor dem "prepared"-Zustand zu bevorzugen, um nicht parallele Subtransaktionen unnötig lange zu sperren, wogegen in lokalen Systemen mit TO-Schedulern die Tickets am Beginn der Subtransaktion modifiziert werden sollten, um synchrone Reihenfolgen von Zeitstempeln und Ticketwerten zu erhalten und so unnötige Transaktionsabbrüche zu vermeiden.

Zusammenfassend läßt sich feststellen, daß die optimistische Ticketmethode globale Serialisierbarkeit gewährleistet, wenn die lokalen Synchronisationsmechanismen lokale

Serialisierbarkeit garantieren, jede globale Transaktion aus maximal einer Subtransaktion pro CDBMS besteht und jede Subtransaktion einen sichtbaren "prepared-to-commit"-Zustand hat. Es werden den CDBMSen vom GTM keine Restriktionen der lokalen Ausführung der Subtransaktionen aufgezwungen, d.h. die Ausführungsautonomie der lokalen Knoten wird gewährleistet. Nachteilig an der OTM sind die möglichen globalen "Restarts", das Verwalten des GSG sowie die zusätzlichen Ticketkonflikte zwischen sonst konfliktfreien globalen Transaktionen.

b) Die Konservative Ticketmethode (CTM)

Wie bereits erwähnt, hat der OTM-Ansatz den Nachteil, daß eine Validierung erst zum "commit"-Zeitpunkt erfolgt, so daß unterschiedliche globale Ticketreihenfolgen zu globalen Restarts führen können. Die *konservative Ticketmethode* (CTM – "conservative ticket method") bezeichnet einen pessimistischen Ansatz, der globale Restarts durch unterschiedliche Ticketreihenfolgen von vornherein ausschliessen soll. Diese Methode erzwingt, daß alle globalen Subtransaktionen ihre Ticketoperationen in den beteiligten Systemen in der gleichen Reihenfolge ausführen und kontrolliert die Reihenfolge der Ticketoperationen.

Dies erfordert zusätzlich zu den bereits für die OTM benötigten Voraussetzungen das Vorhandensein eines sichtbaren "prepared-to-take-a-ticket"-Zustands in den Subtransaktionen. Dieser Zustand kann mit den gleichen Techniken zur Verfügung gestellt werden, mit denen sich auch der "prepared-to-commit"-Zustand simulieren läßt. Eine Subtransaktion tritt in den "prepared-to-take-a-ticket"-Zustand, wenn sie alle Datenbankoperationen ausgeführt hat, die der Ticketoperation vorangehen, und verläßt diesen Zustand, wenn sie den Ticketwert liest. Eine globale Transaktion ist im "prepared-to-take-a-ticket"-Zustand, wenn alle ihre Subtransaktionen es sind.

Die Abarbeitung einer Menge globaler Transaktionen $\mathcal{G}$ mittels CTM läuft wie folgt ab: Der GTM setzt zunächst ein Timeout-Intervall für jede globale Transaktion aus $\mathcal{G}$ und reicht die Subtransaktionen zur lokalen Abarbeitung an die CDBMSe weiter. Diese bearbeiten die Subtransaktionen selbständig, bis der sichtbare "prepared-to-take-a-ticket"-Zustand erreicht wird. Ohne Beschränkung der Allgemeinheit nehmen wir an, daß $G_1, G_2, \ldots, G_k$ aus $\mathcal{G}$ den "prepared-to-take-a-ticket"-Zustand erreichen, bevor deren Timeout-Intervalle abgelaufen sind. Wir nehmen weiterhin an, daß G_1 diesen Zustand vor G_2 erreicht, d.h. mindestens eine Subtransaktion von G_2 erreicht den "prepared-to-take-a-ticket"-Zustand, nachdem alle Subtransaktionen von G_1 ihn erreicht haben, G_2 ist vor G_3 bereit zur Ticketoperation und G_{k-1} vor G_k. Die CTM erlaubt dann allen Subtransaktionen von G_1 ihre Ticketoperationen vor den Subtransaktionen von G_2 auszuführen, G_2 macht die Ticketoperationen vor G_3 und G_{k-1} holt die Tickets vor G_k. Eine globale Transaktion G_i wird mit "commit" beendet, wenn ihre Ticketoperationen erfolgreich abgearbeitet wurden und alle Subtransaktionen den "prepared-to-commit"-Zustand erreicht haben. Dagegen bricht der GTM globale Transaktionen ab und startet diese neu, wenn deren Timeout-Intervall abgelaufen ist.

Die konservative Ticketmethode gewährleistet somit die globale Serialisierbarkeit, wenn die lokalen Synchronisationsmechanismen lokale Serialisierbarkeit garantieren, jede globale Transaktion aus maximal einer Subtransaktion pro CDBMS besteht und jede Subtransaktion sowohl einen sichtbaren "prepared-to-take-a-ticket"-Zustand als auch einen sichtbaren "prepared-to-commit"-Zustand hat. Ein wesentlicher Vorteil der CTM gegenüber der OTM ist der Wegfall des GSG-Behandlung, wodurch der globale Scheduler wesentlich einfacher wird.

4.1.2 Teilungsserialisierbarkeit

Das zweite in [ZE93] vorgestellte Korrektheitskriterium für globale Subtransaktionen realisiert, daß in jedem beteiligten Knoten die Ausführungsreihenfolge der *teilenden* Operationen der globalen Subtransaktionen identisch zur globalen Serialisierungreihenfolge gehalten wird. Dieses Kriterium nennt sich *Teilungsserialisierbarkeit* (SSR) und liefert dem GTM eine ausreichende Bedingung zur Synchronisation der relativen Serialisierungsreihenfolgen der Subtransaktionen aller globalen Transaktionen. Wie schon bei der CCSR sollen zunächst einige Grundbegriffe definiert werden:

Definition 4.3 (Vollständig teilende Transaktionen) Eine Menge lokaler Transaktionen $\mathcal{T}$ ist vollständig teilend, wenn es eine totale Reihenfolge $T_{i_1}, T_{i_2}, \ldots, T_{i_m}$ unter den Transaktionen aus $\mathcal{T}$ derart gibt, daß $D_{T_{i_1}} \subseteq D_{T_{i_2}} \subseteq \ldots \subseteq D_{T_{i_m}}$ gilt. Eine Menge globaler Transaktionen $\mathcal{G}$ ist vollständig teilend, wenn es eine totale Reihenfolge O in $\mathcal{G}$ derart gibt, daß in jedem Knoten s_i $(1 \leq i \leq n)$ die Menge der dort ausgeführten globalen Subtransaktionen in einer Reihenfolge vollständig teilend ist, die konsistent zu O ist. □

Die Transaktionsbeziehung der vollständigen Teilung ist nur im Hinblick auf die Datenmengen der Transaktionen definiert, die Art der Transaktionsoperationen wird nicht berücksichtigt. Es ist intuitiv klar, daß eine Menge von Konfliktkettentransaktionen nicht unbedingt vollständig teilend sein muß, umgekehrt bilden vollständig teilende Transaktionen nicht unbedingt eine Konfliktkette. [ZE93] zeigen, daß in einem serialisierbaren Schedule S für zwei vollständig teilende Transaktionen T_1 und T_2 mit $D_{T_1} \subseteq D_{T_2}$ gilt, daß aus der Ausführungsreihenfolge $o_1 \prec_S o_2$ aller teilender Operationen $o_1 \in OP_{T_1}, o_2 \in OP_{T_2}$ die Serialisierungsreihenfolge $T_1 \rightarrow T_2$ für die teilenden Transaktionen in diesem Schedule folgt.

Definition 4.4 (Teilungsäquivalenz) Zwei globale Subschedules $S_\mathcal{G}$ und $S'_\mathcal{G}$ der Schedules S und S' sind teilungsäquivalent ($S_\mathcal{G} \equiv_t S'_\mathcal{G}$), wenn sie die gleichen Operationen von Transaktionen aus $\mathcal{G}$ enthalten, wobei $\mathcal{G}$ vollständig teilend in der Reihenfolge O ist, und aus der Reihenfolge zweier globaler Transaktionen G_i vor G_j in O folgt, daß in jedem Knoten s_k $(1 \leq k \leq n)$ für alle dort teilenden Operationen $o_{ik} \in OP_{G_{ik}}, o_{jk} \in OP_{G_{jk}}$ die folgenden Ausführungsreihenfolgen gelten: $o_{ik} \prec_{S_\mathcal{G}} o_{jk}$ und $o_{ik} \prec_{S'_\mathcal{G}} o_{jk}$. □

Definition 4.5 (Teilungsserialisierbarkeit) Ein globaler Subschedule $S_{\mathcal{G}}$ ist genau dann teilungsserialisierbar, wenn seine "commit"-abgeschlossene Projektion $C(S_{\mathcal{G}})$ teilungsäquivalent zu einem seriellen globalen Subschedule ist. □

Auch die Teilungsserialisierbarkeit ist eine Verschärfung des allgemeinen Serialisierbarkeitsbegriffs, d.h. jeder teilungsserialisierbare Schedule ist auch serialisierbar, die Umkehrung der Aussage gilt jedoch nicht. In [ZE93] wird folgender Satz bewiesen:

Satz 4.2 Es sei S ein globaler Schedule und $\mathcal{G}$ die Menge der globalen Transaktionen in S. Wenn $S_{\mathcal{G}}$ teilungsserialisierbar ist, dann impliziert die Serialisierbarkeit der lokalen Schedules in den Knoten s_k $(1 \leq k \leq n)$ die globale Serialisierbarkeit von S. □

Folglich läßt sich die Teilungsserialisierbarkeit globaler Subschedules in Multidatenbankumgebungen als hinreichende Bedingung für die Gewährleistung globaler Serialisierbarkeit nutzen. Es ist intuitiv klar, daß der vorgestellte Ansatz ebenso gilt, wenn in den Definitionen und Sätzen anstatt von einer "$\subseteq$"-Beziehung von einer "$\supseteq$"-Beziehung ausgegangen wird. In [ZE93] wird wie für die CCSR auch für die SSR eine Graphentestmethode vorgestellt, in der die Zyklenfreiheit des Graphen eines globalen Schedules dessen SSR-Eigenschaft anzeigt. Eine detaillierte Erläuterung dieser Methode ist den Ausführungen in [ZE93] zu entnehmen. Das folgende Beispiel verdeutlicht statt dessen die Idee des SSR-Ansatzes:

Beispiel 4.3 Ein FDBS bestehe aus zwei Knoten: s_1 mit dem Datenobjekt a, s_2 mit den Datenobjekten b und c. GT_1 und GT_2 seien zwei globale Transaktionen:

$$GT_1 : w_1(a)r_1(b)$$
$$GT_2 : r_2(a)w_2(c)r_2(b)$$

Die globalen Transaktionen sind vollständig teilend in der Reihenfolge $GT_1 \rightarrow GT_2$ (im Knoten s_1 gilt $D_{GT_1} = D_{GT_2} = \{a\}$ und im Knoten s_2 gilt $D_{GT_1} \subset D_{GT_2}$, da $\{b\} \subset \{b, c\}$ ist). Zusätzlich existiert eine lokale Transaktion LT_3 auf dem Knoten s_2:

$$LT_3 : w_3(b)r_3(c)$$

Der globale Schedule S bestehe aus folgenden lokalen Schedules S_1, S_2:

$$S_1 : w_1(a)r_2(a)$$
$$S_2 : w_3(b)r_1(b)r_3(c)w_2(c)r_2(b)$$

$S_{\mathcal{G}}$ ist offensichtlich teilungsserialisierbar in der Reihenfolge $GT_1 \rightarrow GT_2$ und S ist global serialisierbar. S_2 zeigt anschaulich, daß Konflikte zwischen lokalen und globalen Transaktionen keine anderen Serialisierungsreihenfolgen in Knoten mit konfliktfreien globalen Transaktionen erzeugen können, wie es in Beispiel 3.2 der Fall gewesen ist, da jeder Konflikt $LT_3 \rightarrow GT_1$ auch einen Konflikt $LT_3 \rightarrow GT_2$ impliziert, und so durch die lokale Serialisierbarkeit kein Konflikt $GT_2 \rightarrow LT_3$ möglich ist. □

Für die SSR gilt, daß solange die Ausführungsreihenfolge der teilenden Operationen globaler Transaktionen in allen Knoten konstant gehalten wird, kann die globale Serialisierbarkeit durch die lokalen Scheduler nicht verletzt werden. Die bereits erwähnte Zusatzoperationsmethode kann ebenfalls zur Forcierung von globalen Subtransaktionen mit SSR-Eigenschaft genutzt werden, wobei jedoch lediglich "read"-Operationen an nicht vollständig teilende Transaktionen anzuhängen sind. Da "read"-Operationen weniger Blockierungen als "update"-Operationen nach sich ziehen, gilt die SSR als einfacher und effizienter zu realisieren als die CCSR. Jedoch kann die Forcierung der SSR-Eigenschaften in globalen Transaktionen, die überhaupt keine Datenobjekte teilen, zu sehr langen "Anhängseln" führen. Aus diesem Grund werden beim nächsten Kriterium die Ansätze aus CCSR und SSR gemischt.

4.1.3 Hybride Serialisierbarkeit

Als Kreuzung aus Konfliktketten- und Teilungsserialisierbarkeit erbt die *hybride Serialisierbarkeit* wesentliche Eigenschaften der CCSR und SSR. Zunächst sollen wieder die benötigten Grundbegriffe definiert werden:

Definition 4.6 (Hybride Transaktionen) Eine Menge lokaler Transaktionen $\mathcal{T}$ ist hybrid, wenn es eine totale Reihenfolge $T_{i_1}, T_{i_2}, \ldots, T_{i_m}$ unter den Transaktionen aus $\mathcal{T}$ derart gibt, daß $D_{T_{i_1}} \Diamond D_{T_{i_2}} \Diamond \ldots \Diamond D_{T_{i_m}}$ gilt, wobei $\Diamond \in \{\overset{c}{\sim}, \subseteq, \supseteq\}$[2] ist und jeweils drei fortlaufend geordnete Transaktionen $T_{i_j}, T_{i_{j+1}}, T_{i_{j+2}}$ $(1 \leq j \leq n-2)$ nicht via $T_{i_j} \supseteq T_{i_{j+1}} \subseteq T_{i_{j+2}}$ miteinander assoziiert sind. Eine Menge globaler Transaktionen $\mathcal{G}$ ist hybrid, wenn es eine totale Reihenfolge O in $\mathcal{G}$ derart gibt, daß in jedem Knoten s_i $(1 \leq i \leq n)$ die Menge der dort ausgeführten globalen Subtransaktionen in einer Reihenfolge hybrid ist, die konsistent zu O ist. □

Definition 4.7 (Hybride Äquivalenz) Zwei globale Subschedules $S_\mathcal{G}$ und $S'_\mathcal{G}$ der Schedules S und S' sind hybrid äquivalent $(S_\mathcal{G} \equiv_h S'_\mathcal{G})$, wenn sie die gleichen Operationen von von Transaktionen aus $\mathcal{G}$ enthalten, wobei $\mathcal{G}$ hybrid in der Reihenfolge O ist, und aus der Reihenfolge zweier globaler Transaktionen G_i vor G_j in O folgt, daß in jedem Knoten s_k $(1 \leq k \leq n)$ die folgenden Bedingungen gelten:

- Wenn $G_{ik} \overset{c}{\sim} G_{jk}$ in O gilt, dann gelten in s_k für alle Konfliktoperationen $o_{ik} \in OP_{G_{ik}}, o_{jk} \in OP_{G_{jk}}$ die folgenden Ausführungsreihenfolgen: $o_{ik} \prec_{S_\mathcal{G}} o_{jk}$ und $o_{ik} \prec_{S'_\mathcal{G}} o_{jk}$.

[2] Die "$\overset{c}{\sim}$"-Beziehung hat in dieser Menge eine höhere Priorität, wogegen "$\subseteq$" und "$\supseteq$" die gleiche Priorität haben. Sind also zwei Transaktionen T_i und T_j sowohl in Konflikt $(T_i \overset{c}{\sim} T_j)$ als auch vollständig teilend $(T_i \subseteq (\supseteq) T_j)$, so wird die Konfliktbeziehung für die hybride Ordnung benutzt.

- Wenn $G_{ik} \subseteq G_{jk}$ (oder $G_{ik} \supseteq G_{jk}$) in O gilt, dann gelten in s_k für alle teilenden Operationen $o_{ik} \in OP_{G_{ik}}, o_{jk} \in OP_{G_{jk}}$ die folgenden Ausführungsreihenfolgen: $o_{ik} \prec_{S_\mathcal{G}} o_{jk}$ und $o_{ik} \prec_{S'_\mathcal{G}} o_{jk}$.

□

Definition 4.8 (Hybride Serialisierbarkeit) Ein globaler Subschedule $S_\mathcal{G}$ ist genau dann hybrid serialisierbar, wenn seine "commit"-abgeschlossene Projektion $C(S_\mathcal{G})$ hybrid äquivalent zu einem seriellen globalen Subschedule ist. □

Die hybride Serialisierbarkeit ist eine Verschärfung des allgemeinen Serialisierbarkeitsbegriffs und impliziert globale Serialisierbarkeit. Die Operationen der einzelnen Transaktionen, die für die hybriden Eigenschaften globaler Transaktionen verantwortlich sind, werden als hybride Operationen bezeichnet. Es wird in [ZE93] bewiesen, daß wenn die Transaktionen T_1, T_2 und T_3 in einem serialisierbaren Schedule S hybride Eigenschaften haben, d.h. $T_1 \diamondsuit T_2 \diamondsuit T_3$ gilt, wobei $\diamondsuit \in \{\overset{c}{\sim}, \subseteq, \supseteq\}$ ist und $T_1 \supseteq T_2 \subseteq T_3$ nicht erlaubt ist, dann bestimmt die Ausführungsreihenfolge der hybriden Operationen die Serialisierungsreihenfolge der Transaktionen. Das heißt, wenn $T_i \diamondsuit T_{i+1} (1 \leq i \leq 2)$ ist und für alle hybriden Operationen $o_1 \in OP_{T_i}, o_2 \in OP_{T_{i+1}}$ gilt, daß $o_1 \prec_S o_2$, dann folgt daraus die Serialisierungsreihenfolge $T_1 \rightarrow T_2 \rightarrow T_3$, was zu folgendem Satz führt:

Satz 4.3 Es sei S ein globaler Schedule und $\mathcal{G}$ die Menge der globalen Transaktionen in S. Wenn $S_\mathcal{G}$ hybrid serialisierbar ist, dann impliziert die Serialisierbarkeit der lokalen Schedules in den Knoten s_k $(1 \leq k \leq n)$ die globale Serialisierbarkeit von S. □

Der grundlegende Vorteil der hybriden Serialisierbarkeit besteht darin, daß das globale Transaktionsmanagement verschiedene Möglichkeiten zur Verfügung hat, um hybride Reihenfolgen zwischen globalen Transaktionen zu forcieren und so deren lokale Serialisierungsreihenfolgen festzulegen. Zur Erhaltung der globalen Serialisierbarkeit ist es dann für den GTM ausreichend, die Ausführungsreihenfolge der hybriden Operationen zu bestimmen und in allen lokalen Systemen identisch zu halten.

Beispiel 4.4 Ein FDBS bestehe aus zwei Knoten: s_1 mit dem Datenobjekt a, s_2 mit den Datenobjekten b und c. Folgende globale Transaktionen werden abgearbeitet:

$$GT_1 : w_1(a) r_1(b)$$
$$GT_2 : r_2(a) w_2(c) r_2(b)$$
$$GT_3 : r_3(a) r_3(c) r_3(b)$$
$$GT_4 : w_4(a) r_4(c)$$

Die globalen Transaktionen sind hybrid in der Reihenfolge $GT_1 \rightarrow GT_2 \rightarrow GT_3 \rightarrow GT_4$, wobei in s_1 die Abhängigkeit $GT_1 \overset{c}{\sim} GT_2 \subseteq GT_3 \overset{c}{\sim} GT_4$ und in s_2 die Abhängigkeit $GT_1 \subseteq GT_2 \overset{c}{\sim} GT_3 \supseteq GT_4$ gilt. LT_5 sei eine lokale Transaktion auf dem Knoten s_2:

$$LT_5 : w_5(b)r_5(c)$$

Der globale Schedule S bestehe aus folgenden lokalen Schedules S_1, S_2:

$$S_1 : w_1(a)r_2(a)r_3(a)w_4(a)$$
$$S_2 : w_5(b)r_1(b)r_5(c)w_2(c)r_3(c)r_3(b)r_4(c)r_2(b)$$

$S_\mathcal{G}$ ist hybrid serialisierbar in der Reihenfolge $GT_1 \rightarrow GT_2 \rightarrow GT_3 \rightarrow GT_4$ und S ist global serialisierbar. □

Zusammenfassend gilt, daß solange die Ausführungsreihenfolge der hybriden Operationen globaler Transaktionen in allen Knoten konstant und konsistent zur Reihenfolge der hybriden Eigenschaft der globalen Transaktionen gehalten wird, solange kann die globale Serialisierbarkeit durch die lokalen Scheduler nicht verletzt werden. Die Zusatzoperationsmethode kann zur Forcierung von globalen Subtransaktionen mit HSR-Eigenschaft herangezogen werden, sie hat in diesem Fall jedoch wesentlich mehr Optionen bei der Manipulation der globalen Subtransaktionen. So können Konflikt- aber auch teilende Operationen zur Realisierung der HSR-Eigenschaft genutzt werden. Da bei teilenden Subtransaktionen sowohl Unter- wie Obermengenbeziehungen in der hybriden Reihung möglich sind, entstehen bei der Zusatzoperationsmethode typischerweise Phasen mit wachsenden und schrumpfenden "Transaktionsanhängseln".

Die Beziehung der hybriden Serialisierbarkeit zu den vorangegangenen Korrektheitskriterien und zur Menge der durch die OTM realisierbaren Schedules wird in Abbildung 4.1 graphisch dargestellt. Es ist intuitiv klar, daß die Menge der mittels OTM realisierbaren globalen Schedules eine echte Untermenge der CCSR-Schedules ist, da sie nur CCSR-Schedules enthält, die ihre lokalen Konfliktketten immer über denselben lokalen Datenobjekten, den Tickets, bilden. Ebensoleicht läßt sich die Überlappungsbeziehung der CCSR- und SSR-Schedules erkennen. Ein konfliktkettenserialisierbaren (teilungsserialisierbarer) Schedules kann auch teilungsserialisierbar (konfliktkettenserialisierbar) sein, muß es aber natürlich nicht. Aufgrund ihrer flexiblen Definition enthält die Menge der HSR-Schedules die CCSR- und SSR-Schedules als echte Untermengen. Darüber hinaus enthält diese Menge auch Schedules, die weder konfliktketten- noch teilungsserialisierbar sind. Da die hybride Serialisierbarkeit die globale Serialisierbarkeit impliziert, nicht jedoch ihre Voraussetzung ist, ist die Menge der HSR-Schedules eine echte Untermenge der GSR-Schedules.

In [ZE93] wird weiterhin eine *optimale* Bedingung definiert, die es dem GTM erlaubt, indirekt die Serialisierungsreihenfolge globaler Subtransaktionen zu bestimmen, ohne auf zusätzliche Informationen der lokalen Scheduler außer der lokalen Serialisierbarkeit angewiesen zu sein oder den CDBSen Restriktionen aufzuzwingen. Diese optimale Bedingung ist die *extra-hybride Eigenschaft* von Transaktionen, die auch die Abhängigkeit $T_{i_j} \supseteq T_{i_{j+1}} \subseteq T_{i_{j+2}}$ dreier aufeinanderfolgender Transaktionen in einer hybriden Ordnung $T_{i_1}, T_{i_2}, \ldots, T_{i_m}$ gestattet, wenn gleichzeitig $T_{i_j} \subseteq (\supseteq) T_{i_{j+2}}$ gilt. Diese Eigenschaft heißt optimal, weil sie als die schwächste, d.h. am wenigsten restriktive Eigenschaft gilt, die die oben aufgeführten Bedingungen erfüllt.

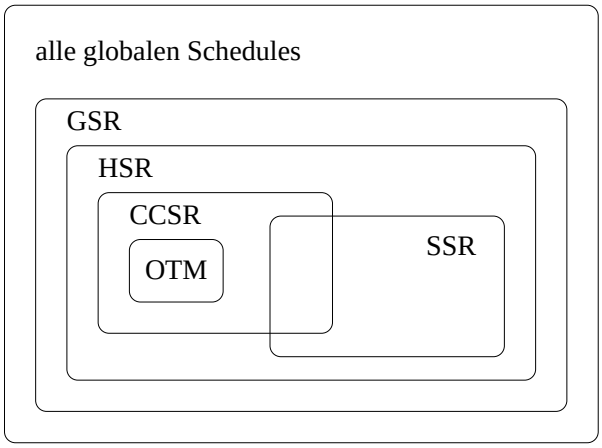

Abbildung 4.1: Beziehungen der vorgestellten Kriterien nach [ZE93]

4.2 DBMSe mit zusätzlichen Eigenschaften

Innerhalb dieses Abschnittes soll nun untersucht werden, inwieweit die Ausnutzung von Wissen über die Transaktionsverwaltungsmechanismen der partizipierenden Datenbanksysteme die Aufgaben des GTM, globale Serialisierbarkeit zu gewährleisten, vereinfachen kann. Dies ist prinzipiell möglich, wie [BGS92, VG93] unterstreichen und man sich leicht am Beispiel 3.2 klarmachen kann. Die Aktionen in den eckigen Klammern zeigen die noch auszuführenden Bearbeitungsschritte:

$$S_1 : r_1(a)c_1w_3(a)w_3(b)c_3r_2(b)[c_2]$$
$$S_2 : w_4(c)r_1(c)c_1r_2(d)[c_2w_4(d)c_4]$$

Gehen wir nun beispielsweise von einem föderierten System aus, in dem alle CDBMSe "Basic TO"-Scheduler (vergleiche Abschnitt 3.1) verwenden, dann tritt im Knoten s_2 folgende Situation ein: Die Transaktionen GT_1, GT_2 und LT_4 erhalten während ihrer Laufzeit zu einem bestimmten Zeitpunkt ihre Zeitstempel ts_1, ts_2 und ts_4. Der "Basic TO"-Algorithmus sichert die lokale Serialisierbarkeit entsprechend der Reihenfolge der Zeitstempel. Da GT_1 vollständig vor dem Beginn von GT_2 abgearbeitet wird, gilt zwangsläufig, daß $ts_1 < ts_2$ ist und GT_1 vor GT_2 serialisiert wird. Folglich kann LT_4 keine Abhängigkeit $GT_2 \rightarrow \ldots \rightarrow GT_1$ erzeugen. Da der Zeitstempel von LT_4 aufgrund des Konfliktes der ersten beiden Operationen des Schedules S_2 kleiner sein muß als der Zeitstempel von GT_1, verletzt die Operation $w_4(d)$ von LT_4 die lokale Serialisierbarkeit, so daß ein Abbruch dieser Transaktion erfolgt. Demnach kann ein Serialisierbarkeitsproblem, wie es im Abschnitt 3.3.1 auftrat, in diesem Fall nicht auftreten.

In einer föderierten Datenbankumgebung, in der alle beteiligten Systeme den "Basic TO"-Algorithmus zur Sicherung lokaler Serialisierbarkeit benutzen, wäre demnach die serielle Ausführung der globalen Transaktionen die einfachste Möglichkeit zur Gewährleistung der globalen Serialisierbarkeit. In den nachfolgenden Abschnitten soll nun untersucht werden, welche lokalen Eigenschaften der GTM nutzen kann, um auch eine

parallele Abarbeitung von globalen Transaktionen zu ermöglichen.

4.2.1 Strenge Serialisierbarkeit in DBMSen

Die *strenge Serialisierbarkeit* (StSR – "strong serializability") lokaler Schedules ist das erste lokale Kriterium, das hier vorgestellt werden soll und das dem GTM Möglichkeiten bietet, globale Serialisierbarkeit zu gewährleisten. [BGS92] definieren die strenge Serialisierbarkeit wie folgt:

Definition 4.9 (Strenge Serialisierbarkeit) Es sei S ein serialisierbarer Schedule. S ist genau dann streng serialisierbar, wenn für zwei Transaktionen T_i und T_j in S die folgende Bedingung gilt: Wenn die letzte Operation von T_i (c_i/a_i) der ersten Operation von T_j vorangeht, dann gibt es einen äquivalenten seriellen Schedule zu S, in dem T_i vor T_j abgearbeitet wird, d.h. T_i wird in S vor T_j serialisiert. □

Die strenge Serialisierbarkeit ist somit die Eigenschaft, die bei "Basic TO"-Mechanismen realisiert wird, wenn die Transaktionen ihren Zeitstempel mit der Ausführung ihrer ersten Operation erhalten[3]. Wie oben gezeigt, kann der GTM in diesem Fall durch die serielle Ausführung der globalen Transaktionen die globale Serialisierbarkeit gewährleisten. Jedoch ist die Forderung nach einer serieller Ausführung recht restriktiv und es gibt viele Situationen, in denen bessere Lösungswege möglich sind. Arbeiten beispielsweise zwei globale Transaktionen auf verschiedenen Knoten, besteht die Notwendigkeit einer seriellen Ausführung nicht.

Mit einem einfachen Knotensperrverfahren [AGS87] kann der GTM die restriktive serielle Abarbeitung verfeinern, indem er Sperren für alle beteiligten Datenbanksysteme verwaltet und die Ausführung globaler Transaktionen erst einleitet, wenn diese für alle involvierten Knoten die Sperren erworben haben. Die Sperren werden von den Transaktionen erst wieder freigegeben, wenn diese vollständig abgearbeitet wurden. Auch dieser Ansatz ist noch zu restriktiv, wie z.B. die Situation zeigt, in der zwei globale Transaktionen GT_1 und GT_2 abgearbeitet werden sollen, wobei GT_1 auf die Knoten s_1 und s_2 zugreift, während GT_2 auf s_2 und s_3 arbeiten möchte. Intuitiv ist klar, daß beide Transaktionen parallel arbeiten können, da aufgrund der lokalen Serialisierbarkeit in s_2 als einzigen gemeinsamen Knoten keine Zyklen im globalen Serialisierbarkeitsgraphen entstehen können. Dieser Ansatz wird in den folgenden beiden Methoden weiter verfeinert.

[3] Ohne diese Annahme kann ein "Basic TO"-Scheduler auch nicht streng serialisierbare Schedules erzeugen. So könnte z.B. der nicht streng serialisierbare Schedule $w_2(x)r_3(x)c_3w_1(y)c_1w_2(y)c_2$ erzeugt werden, wenn es möglich ist, daß T_1 den kleinsten Zeitstempel erhält, obwohl T_3 vollständig vor der ersten Operation von T_1 abgearbeitet wird. In diesem Fall entspricht die Ausführungsreihenfolge nicht der Serialisierungsreihenfolge $T_1 \rightarrow T_2 \rightarrow T_3$.

a) Knotengraphenalgorithmus

Der *Knotengraphenalgorithmus* (*"site graph algorithm"*) wird in [BS88] als Methode des GTM zur Sicherung der globalen Serialisierbarkeit vorgestellt. Der Knotengraphenalgorithmus dient als Synchronisationsmechanismus im "Amoco Distributed Database System" (ADDS) und geht davon aus, daß Subtransaktionen, die zeitgleich auf demselben Datenbankknoten arbeiten, durch lokale Transaktionen jederzeit in indirekten Konflikten zueinander stehen können, auch wenn sie global konfliktfrei sind. Durch diese indirekten Konflikte können lokale Serialisierungsreihenfolgen entstehen, die durch den GTM nicht zu erkennen sind, das heißt, wenn die Subtransaktionen zweier globaler Transaktionen in mindestens zwei Datenbankknoten zeitlich überlappend abgearbeitet werden, kann es passieren, daß die Subtransaktionen in den beiden Knoten unterschiedlich serialisiert werden, auch wenn sie aus globaler Sicht konfliktfrei sind.

Unter der Annahme streng serialisierbarer lokaler Abarbeitungen können solche Situationen verhindert werden, wenn man ausschließt, daß die Subtransaktionen zweier globaler Transaktionen in zwei oder mehreren Datenbankknoten zeitlich überlappend abgearbeitet werden. Dazu verwaltet der GTM einen bipartiten Graphen mit den Transaktionen und Datenbanksystemen als Knotenmenge. Soll eine neue Transaktion T_i abgearbeitet werden, so werden in den Graphen Kanten eingefügt, die den Knoten der Transaktion T_i mit den Knoten der Datenbanksysteme verbindet, auf die T_i zugreifen soll. Tritt dabei kein Zyklus auf, so wird die Transaktion gestartet. Existiert dagegen ein Zyklus, der T_i enthält, so wird die Ausführung von T_i verzögert.

In [BS88] werden die für die Korrektheit des Ansatzes benötigten Voraussetzungen nur unzureichend definiert. In diesem Artikel wird davon ausgegangen, daß der GTM einen globalen Schedule erzeugt, der serialisierbar ist, wobei der globale Schedule in [BS88] nur aus den globalen Transaktionsoperationen besteht. Für die partizipierenden Systeme wird angenommen, daß sie lokal serialisierbare, nicht jedoch streng serialisierbare Schedules produzieren. Aus den Ausführungen in [BS88] wird damit nicht ersichtlich, wie unter den gegebenen Voraussetzungen eine Situation, wie sie das Beispiel 3.2 zum Ausdruck bringt, verhindert werden kann. Unter Annahme streng serialisierbarer lokaler Schedules läßt sich der Knotengraphenalgorithmus jedoch problemlos anwenden.

Die Arbeitsweise des Knotengraphenalgorithmus läßt sich am Beispiel 3.2 anschaulich darstellen. Wird vom GTM die Transaktion GT_1 gestartet, so werden in den Graphen die Kanten (GT_1, s_1) und (GT_1, s_2) eingefügt. Beim Start von GT_2 erzeugen die Kanten (GT_2, s_1) und (GT_2, s_2) jedoch einen Zyklus im Knotengraphen und der GTM verzögert die Transaktion GT_2 so lange, bis GT_1 in allen Knoten vollständig abgearbeitet wurde. Da sich nun GT_1 und GT_2 nicht mehr überlappen und alle lokalen Systeme strenge Serialisierbarkeit realisieren, wird in den beiden Datenbanksystemen GT_1 vor GT_2 serialisiert und so ein Zyklus im globalen Serialisierbarkeitsgraphen vermieden. Würde dagegen GT_2 in den Knoten s_2 und s_3 ausgeführt, so muß GT_2 nicht verzögert werden, da durch den Start dieser Transaktion kein Zyklus im Serialisierbarkeitsgraphen entstehen kann.

Der Knotengraphenalgorithmus wahrt, mit Ausnahme der Restriktion auf strenge Serialisierbarkeit realisierende CDBMSe, die Ausführung- und Kommunikationsautono-

mie lokaler Systeme und beugt korrekt allen möglichen Konfliktzyklen zwischen globalen Transaktionen vor. Er hat jedoch einige entscheidende Nachteile, wie [GRS91, GRS94] ausführen. Der Parallelitätsgrad zwischen globalen Transaktionen ist vergleichsweise gering, da das gleichzeitige Arbeiten von globalen Transaktionen in einem lokalen Knoten stark eingeschränkt wird. Nicht alle Zyklen im Knotengraphen führen notwendigerweise zu nichtserialisierbaren globalen Schedules. Einen weiteren Nachteil bildet auch das Verwalten des Knotengraphen.

In [GRS91, GRS94] wird ebenfalls das Realisieren eines sinnvollen praktischen Ansatzes beim Löschen von Knoten und Kanten aus dem Graphen als schwierig und damit nachteilig beschrieben, jedoch bezieht sich dieses Argument auf die Anwendung des Algorithmus im Sinne von [BS88], d.h. in föderierten Systemen, deren Komponenten nur serialisierbare Schedules produzieren. Der GTM könnte das globale Serialisierbarkeitsproblem aus Beispiel 3.2 dann dadurch lösen, daß er das Löschen des Knotens von GT_1 mit den angrenzenden Kanten nach dem "commit" so lange verzögert, bis alle überlappend abgearbeiteten Transaktionen, in diesem Fall LT_4 im Knoten s_2, ebenfalls mit "commit" beendet wurden. Danach kann GT_2 abgearbeitet werden, ohne die globale Serialisierbarkeit zu gefährden. Dieser Ansatz setzt jedoch voraus, daß der GTM über die Abarbeitung lokaler Transaktionen informiert wird, was der Forderung nach lokaler Autonomie widerspricht. Das Verzögern des Löschens der Knoten schränkt außerdem die Parallelität weiter ein, was diesen Ansatz wenig praktikabel erscheinen läßt. Realisieren dagegen alle lokalen Scheduler streng serialisierbare Abarbeitungen, ist das Löschen der Knoten und Kanten einer globalen Transaktion und damit die Freigabe der lokalen Datenbanksysteme für andere globale Transaktionen schon zum "commit"-Zeitpunkt möglich und stellt somit kein schwierig zu realisierendes Problem dar.

b) Altruistisches Sperren

Das *altruistische Sperren* ist ein weiterer Ansatz zur Verbesserung der Parallelität. In [SGA89] werden altruistische Sperrverfahren im Zuge der Untersuchung von Synchronisationsmechanismen für langlebige Transaktionen (LLT – "long lived transactions") vorgeschlagen, die durch ein vorzeitiges Freigeben von Sperren schon während der Laufzeit der LLT unnötig lange Verzögerungen anderer Transaktionen verhindern und so die Leistungsfähigkeit von Systemen beim Umgang mit LLT erhöhen können.

In [AGS87] wird versucht, einen ähnlichen Ansatz auch in föderierten Datenbanksystemen zu nutzen. Die Basis des altruistischen Sperrens in FDBS-Umgebungen bildet wie beim Knotengraphenalgorithmus das globale Sperren einzelner Datenbankknoten zur Koordination der Ausführung globaler Transaktionen. Zur Verbesserung der Leistungsfähigkeit ermöglicht jedoch das altruistische Sperrprotokoll, die Sperren für einzelne Datenbankknoten schon vor dem Ende einer globalen Transaktion freizugeben.

Zur Illustration dieses Ansatzes stelle man sich eine globale Transaktion GT_1 vor, die nacheinander die Datenbankknoten s_1, s_2 und s_3 durchläuft. Die Transaktion GT_1 arbeitet zunächst im Knoten s_1 und beendet in diesem Knoten alle ihre Aktionen ein-

schließlich der "commit"-Operation, bevor sie mit der Ausführung von Operationen im nächsten Knoten s_2 beginnt. Nachdem sie auch dort ihre Aktionen vollständig abgearbeitet hat, setzt sie ihre Arbeit im Knoten s_3 fort. Mit dem bisherigen Ansatz könnte eine zweite globale Transaktion GT_2, die ebenfalls auf diese drei Datenbankknoten zugreifen möchte, nicht starten, bevor GT_1 vollständig beendet wurde. Beim altruistischen Ansatz ist dies jedoch möglich, wenn die zweite Transaktion *"im Gefolge von"* (*"in the wake of"*) der ersten Transaktion arbeitet. Das heißt, es ist möglich, daß die Transaktion GT_2 auf s_1 zugreifen kann, nachdem GT_1 dort beendet wurde, ebenso auf die Knoten s_2 und s_3. Folglich können sich die zwei Transaktionen nicht in einem Knoten überlappen, auch wenn GT_1 und GT_2 gleichzeitig ausgeführt werden. Da in diesem Beispiel angenommen wird, daß auch GT_2 die Knoten in der Reihenfolge s_1, s_2 und s_3 durchläuft, können aufgrund der strengen Serialisierbarkeit keine zyklischen Abhängigkeiten zwischen GT_1 und GT_2 entstehen, da in jedem Knoten die Subtransaktion von GT_1 stets vollständig vor der Subtransaktion von GT_2 ausgeführt wird.

Bei der Implementierung eines altruistischen Sperrverfahrens gibt es verschiedene Ansätze [AGS87]. Ein einfacher Ansatz ist die bereits angedeutete Verwendung der obigen Knotensperren mit der Ausnahme, daß die Transaktionen die Sperren bereits zu dem Zeitpunkt freigeben, zu dem sie die Arbeit in einem bestimmten Knoten beenden. Jedoch werden die Sperren nicht vollständig freigegeben, sondern verbleiben in einem "markierten" Zustand. Transaktionen, die eine "markierte" Knotensperre anfordern, können diese erhalten, sind jedoch gezwungen, "im Gefolge" der ursprünglichen Transaktion abzulaufen. Zur Realisierung global serialisierbarer Schedules muß der GTM sicherstellen, daß die Relation *"ist im Gefolge von"* zyklenfrei ist. Es muß also verhindert werden, daß wenn beispielsweise eine globale Transaktion GT_1 im Gefolge von GT_2 abläuft, welche wiederum im Gefolge von GT_3 arbeitet, die Transaktion GT_3 auf eine Seite zugreift, die bereits von GT_1 "markiert" freigegeben wurde. Dies kann mit Hilfe eines *Gefolgegraphen* (*"wake-graph"*) realisiert werden, in dem eine Kante zwischen GT_i und GT_j angibt, daß GT_j im Gefolge von GT_i ausgeführt wird.

Wie schon beim Knotengraphenalgorithmus wurde auch beim altruistischen Sperren angenommen, daß die Serialisierbarkeit lokaler Schedules ausreicht, um global serialisierbare Schedules zu erzeugen. [DE89] zeigen jedoch anhand eines einfachen Beispiels, das in seiner Struktur ähnlich dem Beispiel 3.2 angelegt ist, daß diese lokale Eigenschaft nicht ausreicht, um globale Serialisierbarkeit zu gewährleisten. In seiner ursprünglichen Definition sichert das altruistische Sperren lediglich die Quasiserialisierbarkeit (vergleiche Abschnitt 5.3) globaler Abarbeitungen. Nimmt man jedoch an, daß alle lokalen Systeme streng serialisierbare Schedules erzeugen, so gewährleistet das altruistische Sperren global serialisierbare Abarbeitungen.

In [AGS87] werden ebenfalls Recovery-Aspekte betrachtet und es wird der Einsatz von Kompensationstechniken für bereits beendete Subtransaktionen vorgeschlagen, wenn eine globale Transaktion abgebrochen werden muß. Dieser Ansatz steht in enger Beziehung zu dem Konzept der "Sagas", das ebenfalls Kompensationstransaktionen zur Sicherung der Atomarität einsetzt. Da innerhalb dieses Abschnittes von fehler- und ab-

bruchfreien Umgebungen ausgegangen wird, sollen diese Aspekte an dieser Stelle nicht weiter erörtert werden. Eine Erläuterung des Einsatzes von Kompensationstechniken in FDBSen erfolgt im Abschnitt 6.4.

4.2.2 Serialisierungspunktbasierte Kriterien

Die nächste Klasse lokaler Scheduler ist in der Lage, Transaktionen zu verarbeiten, die sogenannte *Serialisierungspunkte* (SP – "serialization point") enthalten. Serialisierungspunkte wurden erstmals von [Pu88] eingeführt und werden von [MRB+92b, DE90] für konkrete Synchronisierungsverfahren genutzt, die im weiteren noch vorgestellt werden sollen. Der SP-Ansatz ist eng mit dem der strengen Serialisierbarkeit verbunden. Ein Serialisierungspunkt ist eine eindeutig bestimmte Aktion im Laufe einer Transaktion, die deren Serialisierungsreihenfolge in einem Schedule festlegt. So ist beispielsweise in einem Zeitstempelscheduler der Zeitpunkt, in dem eine Transaktion ihren Zeitstempel erhält, der Serialisierungspunkt der Transaktion, wogegen in 2PL-Verfahren der Serialisierungspunkt durch die Operation gegeben ist, die die erste Sperrenfreigabe veranlaßt. [MRB+92b] weisen darauf hin, daß der GTM in Knoten, in denen die Scheduler keine Serialisierungspunkte verarbeiten, wie z.B. Serialisierbarkeitsgraphentester, diese Punkte beispielsweise durch die Anwendung von Ticketmethoden [GRS94] simulieren kann. Folgende Definition aus [BGS92] formalisiert den SP-Ansatz:

Definition 4.10 (SP-Schedule) Es sei S ein serialisierbarer Schedule bestehend aus den Transaktionen $T_1, T_2, \ldots, T_n$. S ist genau dann ein SP-Schedule, wenn es eine Abbildung sp der Transaktionen auf einzelne Transaktionsoperationen derart gibt, daß folgende Bedingungen gelten:

1. $sp(T_i) = o_k$ mit $o_k \in OP_{T_i}$ und
2. Wenn $sp(T_i)$ in S vor $sp(T_j)$ abgearbeitet wird, so existiert ein äquivalenter serieller Schedule, in dem T_i vor T_j ausgeführt wird.

□

Die Reihenfolge der Serialisierungspunkte der Transaktionen bestimmt folglich deren Serialisierungsreihenfolge, d.h. geht in einem Schedule der Serialisierungspunkt $sp(T_i)$ dem Serialisierungspunkt $sp(T_j)$ voran, so ist eine Abhängigkeit der Form $T_j \rightarrow \ldots \rightarrow T_i$ im lokalen Serialisierbarkeitsgraphen von S nicht erlaubt.

Die Klasse der SP-Scheduler ist eine echte Untermenge der Scheduler, die streng serialisierbare Abarbeitungen garantieren, wie [BGS92] zeigen. Ist dem GTM bekannt, welche Aktionen der globalen Transaktionen in den verschiedenen Datenbanksystemen die Serialisierungspunkte bilden, kann dieses Wissen zur Verbesserung der Parallelität der Abarbeitung und des Transaktionsdurchsatzes genutzt werden. Wie serialisierungspunktbasierte Ansätze konkret in FDBS-Umgebungen umgesetzt werden können, soll die Beschreibung der folgenden Methoden zeigen.

a) Synchronisation mit vordefinierter Reihenfolge

Die Ausnutzung von Serialisierungspunkten in lokalen Datenbanksystemen ist Inhalt der in [DE90] gemachten Ausführungen. Dort wird ganz allgemein ein Top-Down-Ansatz vorgeschlagen, der die Synchronisation globaler Transaktionen in einer vorgeschriebenen Reihenfolge realisieren soll. [DE90] gehen dabei synchron zu unserem Transaktionsmodell von einem hierarchischen Transaktionsmanagement aus, in dem eine globale Transaktionsverwaltung als eine übergelagerte Schicht oberhalb diverser lokaler Transaktionsverwaltungssysteme residiert.

In einem solchen hierarchischen Modell kann man zwei prinzipielle Vorgehensweisen unterscheiden: Im *Bottom-Up-Ansatz* bestimmen die lokalen Systeme selbständig die jeweiligen Serialisierungsreihenfolgen und der GTM entdeckt Inkompatibilitäten und löst diese geeignet auf. Beim *Top-Down-Ansatz* dagegen wird vom GTM vor dem Weiterleiten der globalen Transaktionen deren globale Serialisierungsreihenfolge festgelegt, und die partizipierenden DBMSe sind für deren lokale Umsetzung verantwortlich. Es erfolgt demnach lokal eine *Synchronisation mit vordefinierter Reihenfolge*.

Die Aufgabe des globalen Transaktionsmanagers bei einem Top-Down-Ansatz besteht also darin, eine geeignete, lokal möglichst leicht zu realisierende Serialisierungsreihenfolge O globaler Transaktionen zu finden und durch Anwendung spezieller Techniken sicherzustellen, daß diese Reihenfolge auch in allen beteiligten Datenbanksystemen umgesetzt wird. [DE90] sprechen in diesem Zusammenhang von *"O-Serialisierbarkeit"*, bei der eine lokale Ausführungsfolge äquivalent zu einer seriellen Ausführungsfolge sein muß, in der die Reihenfolge der globalen Transaktionen kompatibel zu O ist. Sind alle lokalen Ausführungen O-serialisierbar, so ergibt sich daraus die globale Serialisierbarkeit der Transaktionen in einer zu O kompatiblen Reihenfolge.

Die Techniken zur Realisierung O-serialisierbarer Ausführungsfolgen unterscheiden [DE90] in Techniken zur Kontrolle der lokalen Abarbeitung globaler Transaktionen und Techniken zur Kontrolle der Einreichung globaler Transaktionsoperationen. Die erste Gruppe erfordert lokale DBMSe mit wenig bzw. gar nicht restriktiven Autonomieanforderungen, die die Möglichkeit bieten, die lokalen Scheduler zu modifizieren oder Abarbeitungseigenschaften zu ändern. [DE90] beschreiben beispielsweise, wie man 2PL- und SGT-Scheduler so modifizieren kann, daß sie O-Serialisierbarkeit gewährleisten. Auf diese Gruppe von Ansätzen soll hier aufgrund unserer restriktiveren Autonomieanforderungen nicht näher eingegangen werden.

Die zweite Gruppe ist für DBMSe mit wesentlich restriktiveren Autonomieanforderungen geeignet und soll hier näher betrachtet werden. Die grundlegende Idee ist wie bei vorangegangenen Ansätzen, daß durch eine Kontrolle der Reihenfolge des Einreichens bestimmter Operationen bei den lokalen Datenbankmanagementsystemen bestimmte Ausführungsanforderungen, hier O-Serialisierbarkeit, sichergestellt werden können. [DE90] definieren dazu eine Architektur, die dem Transaktionsmodell des Abschnitts 3.2 entspricht. Im Gegensatz zu bisherigen Ansätzen wird hier jedoch das Wissen über lokale Synchronisationsmechanismen in die Serverprozesse gepackt, die *"Stub-Prozesse"* genannt und individuell für jedes DBMS implementiert werden. So wird ein wesentlicher

Teil der Komplexität der globalen Transaktionsverwaltung vom GTM in die Server weitergereicht. Gleichzeitig ist es möglich, auf unterschiedliche Autonomiebedürfnisse lokaler DBMSe individuell zu reagieren.

Der GTM bestimmt nur noch eine globale Serialisierungsreihenfolge O und gibt die einzelnen Subtransaktionen und die ermittelte Reihenfolge zur Umsetzung an die "Stub-Prozesse" weiter. Diese müssen nun in Abhängigkeit von den zugrundeliegenden lokalen Transaktionsverwaltungsmechanismen geeignete Techniken anwenden, um die Reihenfolge O durchzusetzen. Ein "Stub-Prozeß" ist somit eine Funktion mit einer Menge von Subtransaktionen und einer linearen Ordnung darüber als Eingabe und einer Einreichungsstrategie als Ergebnis. Bei der Erstellung einer solchen Strategie nutzt der "Stub-Prozeß" im wesentlichen sein Wissen über die Serialisierungspunkte der globalen Subtransaktionen in dem entsprechenden lokalen DBMS. Er reicht dabei die einzelnen Subtransaktionen in der vorgegebenen Reihenfolge an die lokale Transaktionsverwaltung weiter, jedoch wird eine Subtransaktion erst eingereicht, nachdem die vorangegangene ihren Serialisierungspunkt abgearbeitet hat. Dazu ist es nötig, daß eine Kommunikation mit dem DBMS stattfindet, z.B. durch "Acknowledge-Signale", um den Fortschritt der Abarbeitung einer Subtransaktion dem Server bekannt zu machen. Da dem "Stub-Prozeß" die Serialisationspunkte der Transaktionen bekannt sind, kann er durch Einfügen von Kommunikationsereignissen an diesen Stellen eine Kommunikation mit dem CDBS forcieren.

[DE90] weisen darauf hin, daß der Top-Down-Ansatz globale Verklemmungsfreiheit gewährleistet, da die Serialisierungsreihenfolge der globalen Transaktionen vor dem Einreichen bestimmt wird und so eine globale Transaktion bloß auf Sperren warten kann, die von Transaktionen gehalten werden, die vor der entsprechenden Transaktion serialisiert wurden. Der vorgestellte "Stub-Prozeß"-Ansatz wurde in ein Prototypsystem des InterBase-Labors der Purdue-Universität integriert. Das System vereinigt verschiedene DBMSe, wie Ingres, Guru und Dbase IV, und andere Softwaretools, wie Bourneshell und Awk, die auf verschiedenen Hardwareplattformen laufen können, z.B. Sun, Sequent, IBM/pc oder Macintosh. Der Ansatz basiert auf einem speziellen Kommunikations- und Datenaustausch-Protokoll, das lokale "Access Manager" als speziellen Typ von "Stub-Prozessen" benutzt, jedoch noch keine globalen Transaktionsmechanismen, z.B. zur Synchronisation von globalen Transaktionen, zur Verfügung stellt.

b) Der Transaktion-Knoten-Graphen-Algorithmus

Der nachfolgende Ansatz stammt aus [MRB+92b] und beschreibt sehr detailliert eine Implementationsmöglichkeit für einen globalen Transaktionsmanager. [MRB+92b] arbeiten dabei mit dem im Abschnitt 3.2 vorgestellten Transaktionsmodell. In diesem Artikel wird ein Ansatz gewählt, der das Problem der Sicherung der globalen Serialisierbarkeit von Transaktionen auf die serialisierbare Ausführung eines Schedules reduziert, wie er in lokalen, zentralisierten Datenbanksystemen auftritt.

Dazu wird zunächst für jede globale Transaktion GT_i eine Transaktion $\hat{G}_i$ definiert, die alle Serialisierungspunkte $sp_k(GT_i)$, die hier Serialisierungsfunktionen genannt werden, von GT_i in den einzelnen Knoten s_k $(1 \leq k \leq m)$ enthält. Für einen globalen Schedule S ist dann $ser(S)$ ein Schedule, der die Operationen aller $\hat{G}_i$ in einer zu S konsistenten Reihenfolge enthält. Der Schedule $ser(S)$ ist demnach eine Restriktion des Schedules S, jedoch werden in $ser(S)$ entsprechend der Bedeutung der Operationen Konflikte anders definiert. Zwei Operationen $sp_k(GT_i)$ und $sp_l(GT_j)$ aus $ser(S)$ sind genau dann in einem Konflikt, wenn $k = l$ ist, d.h. GT_i und GT_j wollen auf denselben Knoten zugreifen.

Besteht nun eine föderierte Datenbankumgebung ausschließlich aus Systemen, die SP-Schedules realisieren, so folgt aus der Serialisierbarkeit von $ser(S)$ die globale Serialisierbarkeit der Transaktionen im FDBS. Der Schedule $ser(S)$ ist nämlich genau dann serialisierbar, wenn in allen gemeinsamen Knoten die Serialisierungsfunktionen zweier Transaktionen in der gleichen Reihenfolge ausgeführt werden. Somit läßt sich das Problem der Sicherung globaler Serialisierbarkeit auf das Problem der Sicherung eines einfachen serialisierbaren Schedules reduzieren. Demnach reicht es aus, eine Methode zu finden, die die Serialisierbarkeit von $ser(S)$ gewährleistet.

Um das zu erreichen, unterteilen [MRB+92b] den GTM in zwei Hauptkomponenten, was in Abbildung 4.2 dargestellt wird. Der GTM_1 nutzt dabei das Wissen über die lokalen Synchronisationsmechanismen, um für jede Transaktion GT_i die Serialisierungsfunktionen $sp_k(GT_i)$ $(1 \leq k \leq m)$ zu bestimmen und schickt diese zur Verarbeitung an den GTM_2. Alle weiteren globalen Datenbankoperationen werden direkt an die Server und lokalen DBMSe weitergereicht. Dabei wird eine Operation (außer der ersten) erst dann an den Server oder den GTM_2 geschickt, wenn die Ausführung der vorangegangenen Operation dem GTM_1 bestätigt wurde. Der GTM_2 ist nur dafür verantwortlich, daß die an ihn gesendeten Operationen in einer serialisierbaren Ausführungsfolge abgearbeitet werden, d.h. er realisiert die Serialisierbarkeit von $ser(S)$.

Zur Durchsetzung der Serialisierbarkeit von $ser(S)$ können prinzipiell alle Verfahren verwendet werden, die man für die klassische Transaktionverwaltung in zentralen DBMSen einsetzt. Da jedoch die Zahl der globalen Transaktionen in einem FDBS im Vergleich zu der Knotenanzahl sehr groß ist und dadurch $ser(S)$ eine große Anzahl von Konflikten enthält, sollte ein konservativer Ansatz mit hoher Parallelität verwendet werden. In [MRB+92b] werden vier solche Ansätze vorgestellt, von denen hier einer näher erläutert werden soll. Alle vier Ansätze sind dabei recht ausführlich und detailliert dargestellt worden und bieten somit eine gute Implementierungsgrundlage. Beispielhaft soll hier das "Schema 1" aus [MRB+92b] in allen seinen Details vorgestellt werden, um dem Leser einen Eindruck von der Komplexität des Ansatzes zu vermitteln.

Da die Umsetzung der Serialisierbarkeit von $ser(S)$ durch den GTM_2 zu realisieren ist, soll zunächst die Grundstruktur dieser Komponente erläutert werden, die in Abbildung 4.3 dargestellt ist. Wie bereits geschildert, werden vom GTM_1 alle Operationen $sp_k(GT_i)$ einer globalen Transaktion GT_i zur Verarbeitung an den GTM_2 weitergereicht. Alle Serialisierungsfunktionen einer globalen Transaktion GT_i werden vom GTM_1 zusätz-

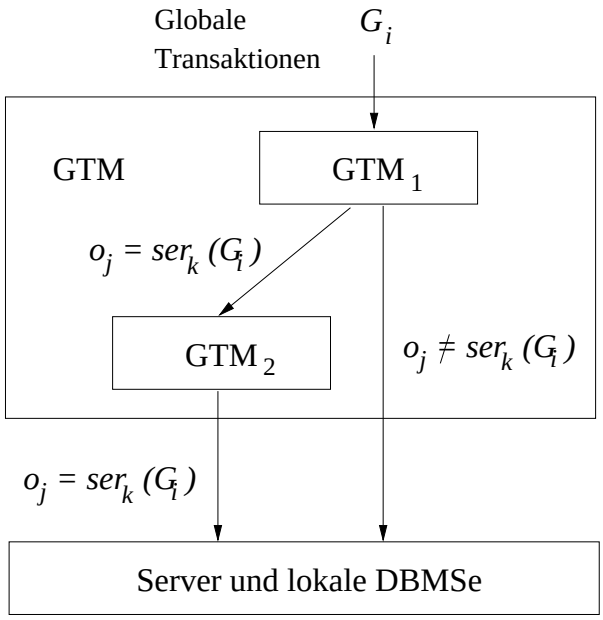

Abbildung 4.2: Die GTM-Komponenten [MRB+92b]

lich mit $init_i$ und fin_i geklammert, so daß sie Transaktionseigenschaften bekommen ($\hat{G}_i$). Die Operationen werden im GTM_2 in einer Warteschlange ("Queue") in der Reihenfolge ihres Eintreffens zwischengespeichert. Die Synchronisationskomponente (CC – "concurrency control component") entnimmt die einzelnen Operationen o_i der Warteschlange und prüft die Bedingungen $cond(o_i)$, die Auskunft gibt, ob die mit der Operation verbundene interne Aktionsfolge $act(o_i)$ abgearbeitet werden kann oder die Ausführung der Operation verzögert werden muß. Dazu werden von der Synchronisationskomponente zusätzliche Datenstrukturen (DS – "data structure") verwaltet.

Die vier in [MRB+92b] vorgestellten Ansätze unterscheiden sich nun in den zu verwaltenden Datenstrukturen DS, sowie den Operationsbedingungen und internen Aktionen. Hier soll beispielhaft der *Transaktion-Knoten-Graphen-Algorithmus* vorgestellt werden, der als Datenstruktur einen Graphen ähnlich dem Knotengraphen aus [BS88] verwaltet. Der *Transaktion-Knoten-Graph* (TSG – "transaction site graph") ist ein ungerichteter bipartiter Graph, in dem die einzelnen Datenbanksysteme und die Transaktionen aus $ser(S)$ die Knotenmenge bilden. Der TSG enthält eine Kante zwischen einem Transaktionsknoten und einem Datenbankknoten $(\hat{G}_i, s_k)$, wenn $sp_k(GT_i) \in \hat{G}_i$ ist. Die Synchronisationskomponente verwaltet außerdem für jedes partizipierende Datenbanksystem eine interne Einfüge- und eine Löschwarteschlange (*"insert queue"* bzw. *"delete queue"*).

Zu Beginn der Abarbeitung sind sowohl der Graph als auch die internen Warteschlangen leer. Die einzelnen Operationsbedingungen und Aktionsfolgen sind wie folgt definiert:

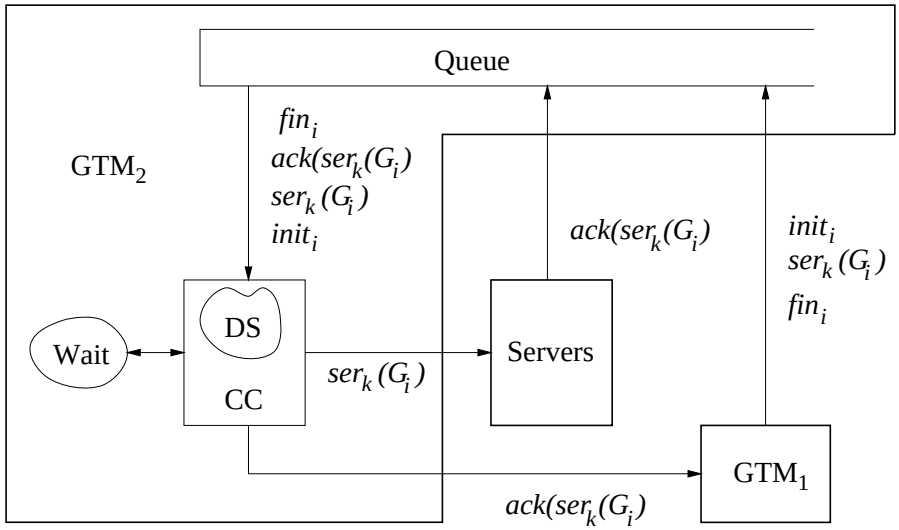

Abbildung 4.3: Die Basisstruktur des GTM_2 [MRB+92b]

- $\boldsymbol{cond(init_i)}$: wahr.

- $\boldsymbol{act(init_i)}$: $\hat{G}_i$ und die zugehörigen Kanten werden in den TSG eingefügt. Außerdem wird für jede Operation $sp_k(GT_i) \in \hat{G}_i$ die Operation an das Ende der "insert queue" des entsprechenden Datenbankknotens gehängt. Hat das Einfügen von $\hat{G}_i$ und der zugehörigen Kanten im TSG einen Zyklus erzeugt, so werden alle Operationen $sp_k(GT_i)$ in den Einfügewarteschlangen zusätzlich markiert.

- $\boldsymbol{cond(sp_k(GT_i))}$: Von jeder Transaktion $\hat{G}_j$ mit $sp_k(GT_j) \in \hat{G}_j$, von der $act(sp_k(GT_j))$ schon ausgeführt wurde, muß auch $act(ack(sp_k(GT_j)))$ schon ausgeführt worden sein. Ist $sp_k(GT_i)$ zusätzlich markiert, so muß $sp_k(GT_i)$ die erste Position in der "insert queue" haben.
  Das bedeutet, daß alle vor $sp_k(GT_i)$ begonnenen Serialisierungsfunktionen auf s_k zum Abarbeitungszeitpunkt vollständig ausgeführt sein müssen. Ist $\hat{G}_i$ zusätzlich in einem Zyklus, so müssen alle globalen Transaktionen, die zuvor ein $act(init_j)$ ausgeführt haben, ihre Serialisierungsfunktionen vollständig abgearbeitet haben.

- $\boldsymbol{act(sp_k(GT_i))}$: Die Operation $sp_k(GT_i)$ wird zur Ausführung an den entsprechenden Server weitergereicht.

- $\boldsymbol{cond(ack(sp_k(GT_i)))}$: wahr.

- $\boldsymbol{act(ack(sp_k(GT_i)))}$: Die Operation $sp_k(GT_i)$ wird aus der Einfügewarteschlange des Knotens s_k gelöscht (die Operation $sp_k(GT_i)$ steht dabei nicht unbedingt am Anfang der "insert queue") und wird an das Ende der "delete queue" des Knotens angehängt. Die Operation $ack(sp_k(GT_i))$ wird an den GTM$_1$ weitergereicht.

- ***cond*(fin_i)** : Jede Operation $sp_k(GT_i) \in \hat{G}_i$ muß die erste Operation in der Löschwarteschlange des entsprechenden Knotens s_k sein.
- ***act*(fin_i)** : $\hat{G}_i$ und die dazugehörigen Kanten werden aus dem TSG gelöscht. Jede Operation $sp_k(GT_i) \in \hat{G}_i$ wird aus der "delete queue" des entsprechenden Knotens gelöscht.

Das hier dargestellte Schema sichert die Serialisierbarkeit von $ser(S)$. Interessant ist, daß dabei entgegen dem Knotengraphenansatz [BS88] Zyklen im TSG erlaubt sind. Jedoch werden Zyklen im Serialisierbarkeitsgraphen von $ser(S)$ vermieden, indem Operationen, deren Ausführung potentiell Zyklen im Serialisierbarkeitsgraphen von $ser(S)$ erzeugen könnten, markiert und dadurch mit zusätzlichen Ausführungsbedingungen belegt werden.

In [MRB+92b] werden drei weitere Schemata vorgestellt, die sich vom obigen Ansatz durch die verwalteten Datenstrukturen, die Ausführungsbedingungen und die internen Aktionsfolgen unterscheiden. Ein Ansatz weist dabei eine geringere Komplexität und Parallelität als das Transaktion-Knoten-Graphen-Schema auf, während die beiden anderen mit steigender Komplexität auch eine höhere Parallelität aufweisen. Auf eine ausführliche Darstellung dieser Ansätze soll hier verzichtet werden, da sie im Vorgehen synchron zum vorgestellten Transaktion-Knoten-Graphen-Algorithmus laufen und demnach keine wesentlichen neuen Aspekte enthalten.

4.2.3 Strenge Rücksetzbarkeit in DBMSen

Die nächste hier vorgestellte Scheduler-Klasse soll die Eigenschaft besitzen, daß die von ihnen erzeugten Ablaufpläne dem Kriterium der *strengen Rücksetzbarkeit* (StRC – "strong recoverability") genügen. Die Idee der strengen Rücksetzbarkeit beruht auf einer Einschränkung des vorherigen Ansatzes. Wenn man fordert, daß die Serialisierungspunkte am Ende der einzelnen Transaktionen erscheinen, d.h. während der "commit"-Operation, dann erhält man dadurch GTM-Mechanismen, die effizienter arbeiten als die der streng serialisierbaren Schedules. Allgemein definiert man strenge Rücksetzbarkeit wie folgt [BGRS91, BGS92, GRS94]:

Definition 4.11 (Strenge Rücksetzbarkeit) Ein Schedule S_k ist *streng rücksetzbar*, wenn für alle Transaktionspaare T_i und T_j folgende Bedingung gilt: Wenn T_i in S_k in einem direkten Konflikt zu T_j steht und T_j endet in S_k mit "commit" (c_j), dann wird c_j nicht ausgeführt, bevor nicht c_i ausgeführt wurde. □

Die strenge Rücksetzbarkeit stellt eine Verschärfung der allgemeinen RC-Eigenschaft [BHG87] dar, sichert jedoch weder die ACA- noch die ST-Eigenschaft eines lokalen Schedules. Während die allgemeine RC-Eigenschaft eine "commit"-Sortierung nur für Transaktionen fordert, die miteinander in einem *"liest-von"*-Konflikt, d.h. einer Untermenge der wr-Konflikte, stehen, müssen bei der StRC-Eigenschaft die "commit"-Operationen

aller in Konflikte involvierten Transaktionen in Konfliktreihenfolge geordnet sein, d.h. es werden sowohl alle wr-Konflikte als auch alle ww- und rw-Konflikte betrachtet. Die Menge der StRC-Schedules ist eine echte Untermenge der RC-Schedules und impliziert, im Gegensatz zur Obermenge, Serialisierbarkeit (vergleiche Abbildung 4.5). Gleichzeitig ist die Menge der StRC-Schedules auch Untermenge der SP-Schedules, wie [BGS92] zeigen.

Wenn alle beteiligten Systeme eines föderierten Datenbanksystems lokale Transaktionsverarbeitungsmechanismen verwenden, die strenge Rücksetzbarkeit garantieren, kann der GTM dieses Wissen zur Realisierung von Abarbeitungsfolgen nutzen, die sich durch eine höhere Parallelität auszeichnen. Die vorangegangenen Ansätze gingen von streng serialisierenden und SP-basierten lokalen Synchronisationsmechanismen aus und vermieden Zyklen im globalen Serialisierbarkeitsgraphen im wesentlichen dadurch, daß sie sicherstellten, daß sich globale Transaktionen nicht oder nur teilweise überlappen. Mit der strengen Rücksetzbarbarkeit ist es dagegen für den GTM ausreichend, sicherzustellen, daß die globalen Transaktionen ihre "commit"-Operationen seriell abarbeiten, d.h. zwischen dem ersten "commit" einer globalen Transaktion G_i auf dem Knoten s_k und dem letzten "commit" von G_i auf s_l darf keine andere globale Transaktion G_j eine "commit"-Operation ausführen.

Dies läßt sich anhand einer modifizierten Version vom Beispiel 3.2 erklären. Die Operationen in Klammern stellen wieder die zukünftigen Aktionen dar:

$$S_1 : r_1(a)c_1w_3(a)w_3(b)c_3r_2(b)[c_2]$$
$$S_2 : w_4(c)r_1(c)r_2(d)r_4(e)c_1c_2[w_4(d)c_4]$$

In diesem Beispiel sind die Aktionen von GT_1 und GT_2 in s_2 überlappend, GT_2 liest d, bevor GT_1 endet. Würde Knoten s_2 nun streng serialisierbare oder SP-Schedules erlauben, könnten die nachfolgenden Aktionen in s_2 ausgeführt werden, ohne die lokalen Serialisierbarkeitskriterien zu verletzen, was jedoch zu einer Verletzung der globalen Serialisierbarkeit führen würde: Der Serialisierbarkeitsgraph von S_1 ist $GT_1 \rightarrow LT_3 \rightarrow GT_2$, während der Graph von S_2 die Abhängigkeit $[GT_2 \rightarrow]LT_4 \rightarrow GT_1$ aufweist. Realisiert jedoch s_2 streng rücksetzbare Schedules, ist diese Abhängigkeit nicht erlaubt, da $GT_2 \rightarrow LT_4 \rightarrow GT_1$ impliziert, daß $c_2 \prec_{s_2} c_4 \prec_{s_2} c_1$ gilt, was aber unmöglich ist, da in dem Schedule schon $c_1 \prec_{s_2} c_2$ gilt. Folglich würden die Transaktionsverwaltungsmechanismen von s_2 die Transaktion LT_4 abbrechen.

Lokale Scheduler, die StRC-Eigenschaften realisieren, geben somit dem GTM die Kontrolle über die Serialisierungspunkte. Das heißt, wenn der GTM die "commit"-Operation einer globalen Transaktion an einen lokalen Scheduler mit StRC-Eigenschaften zur Ausführung übergibt, kann er sicher sein, daß diese globale Transaktion in der globalen Serialisierungsreihenfolge vor jeder anderen globalen Transaktion stehen wird, deren "commit" später ausgeführt werden soll.

a) Commitment Ordering

Eine ausführliche Untersuchung der Möglichkeiten zur Sortierung der "commit"-Operationen zur Realisierung bestimmter Scheduleeigenschaften findet sich auch in [Raz92]. Die Eigenschaft der strengen Rücksetzbarkeit wird in [Raz92] als die CO-Eigenschaft (*"commitment ordering"*) von Ausführungen bezeichnet, die Definition ist aber äquivalent zur Definition 4.11.

In [Raz92] wird gezeigt, wie man mit einfachen Mitteln die CO-Eigenschaft lokaler Schedules realisieren kann. Dazu wird zunächst ein Scheduler modelliert, der, entsprechend der unterschiedlichen Abarbeitungsphasen einer Transaktion, aus zwei Komponenten besteht:

- Der "Resource Access Scheduler" (RAS) behandelt die eintreffenden Datenzugriffsoperationen der Transaktionen und entscheidet, welche Operation wann ausgeführt wird.
- Der "Transaction Termination Scheduler" (TTS) überwacht die Transaktionsmenge und entscheidet, welche Transaktion wann mit "commit" oder "abort" beendet wird. In föderierten Umgebungen nimmt diese Komponente an einem AC-Protokoll teil, um die Atomarität der "commit"-Operationen globaler Transaktionen zu gewährleisten.

In [Raz92] werden zwei "Transaction Termination Scheduler" vorgestellt, die die CO-Eigenschaft lokaler Schedules realisieren und somit eine gute Voraussetzung für ein globales Transaktionsmanagement liefern. Der COCO (*"commitment ordering coordinator"*) realisiert lokale CO-Schedules durch das Verwalten eines Serialisierbarkeitsgraphen (USG – *"undecided serialization graph"*) für nicht beendete (*"undecided"*) Transaktionen. Die Knotenmenge ist die Menge der aktiven Transaktionen und eine Kante $T_i \rightarrow T_j$ ist dann in dem Graphen, wenn T_i in einem Konflikt zu T_j steht. Soll nun eine Transaktion T mit "commit" beendet werden, bestimmt der Scheduler dazu die Menge der abzubrechenden Transaktionen:

$$ABORT_{CO}(T) = \{T' \mid \text{Der USG enthält die Kante } T' \rightarrow T\}$$

Das Abbrechen der in dieser Menge enthaltenen Transaktionen ist notwendige Bedingung zur Realisierung der CO-Eigenschaft des Schedules. Das folgende Beispiel demonstriert die Arbeit mit dem USG:

Beispiel 4.5 Gegeben sei der USG aus Abbildung 4.4. Soll nun T_5 mit "commit" beendet werden, so werden die Transaktionen $ABORT_{CO}(T_5) = \{T_3, T_4\}$ vom COCO abgebrochen und anschließend die Knoten und Kanten von T_3, T_4 und T_5 aus dem USG entfernt. □

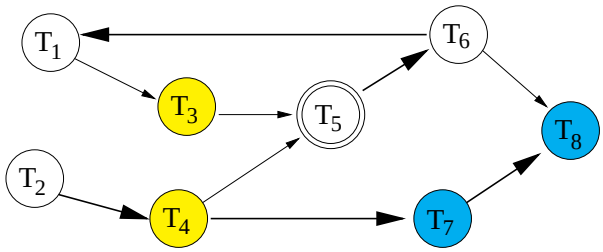

Abbildung 4.4: Beispiel-USG

In [Raz92] wird bewiesen, daß lokale Scheduler, in die ein COCO integriert ist, die CO-Eigenschaft und damit die Serialisierbarkeit der lokalen Ausführung sichern.

Der CORCO (*"commitment ordering recovery coordinator"*) ist der zweite vorgestellte TTS und realisiert neben der CO- auch die RC-Eigenschaft der lokalen Schedules. Dazu wird der obige USG zu einem *wrf*-USG (*"write-read-from"*-USG) erweitert, der die *"liest-von"*-Konflikte gesondert betrachtet. Die Knotenmenge ist wieder die Menge der aktiven Transaktionen. Es werden jedoch zwei disjunkte Kantenmengen verwaltet: Die Kantenmenge C_{wrf} enthält alle die Konflikte, bei denen eine Transaktion T_i einen Datenwert x von einer Transaktion T_j liest. Die Kantenmenge C enthält alle weiteren Konflikte, d.h. sämtliche $ww-$ und $rw-$Konflikte einer Abarbeitung und jene $wr-$Konflikte, die noch nicht in C_{wrf} enthalten sind. Soll nun eine Transaktion T aus dem *wrf*-USG mit "commit" beendet werden, bestimmt der Scheduler zunächst die Menge der abzubrechenden Transaktionen:

$$ABORT_{CO}(T) = \{T' \mid C \text{ oder } C_{wrf} \text{ enthalten die Kante } T' \to T\}$$

Das Abbrechen der in dieser Menge enthaltenen Transaktionen ist notwendig zur Realisierung der CO-Eigenschaft des Schedules. Da aber auch die Rücksetzbarkeit des Schedules realisiert werden soll, ist es nötig, alle Transaktionen, die von den abgebrochenen Transaktionen gelesen haben, ebenfalls abzubrechen:

$$ABORT_{RC}(T') = \{T'' \mid T' \to T'' \text{ ist in } C_{wrf} \text{ oder } T''' \to T'' \text{ ist in } C_{wrf} \text{ mit } T''' \in ABORT_{RC}(T')\}$$

Das Abbrechen der in beiden Mengen enthaltenen Transaktionen ist notwendig zur Realisierung der CO- und RC-Eigenschaft des lokalen Schedules. Das folgende Beispiel demonstriert die Arbeit mit dem *wrf*-USG:

Beispiel 4.6 Gegeben sei als *wrf*-USG der Graph aus Abbildung 4.4, wobei die dicken Kanten die Elemente der Menge C_{wrf} sind, wogegen dünne Kanten $C-$Kanten sind. Soll nun T_5 mit "commit" beendet werden, so werden die Transaktionen in $ABORT_{CO}(T_5) = \{T_3, T_4\}$ zur Realisierung der CO-Eigenschaft abgebrochen. Zur Realisierung der RC-Eigenschaft werden außerdem $ABORT_{RC}(T_3) = \{\emptyset\}$ und $ABORT_{RC}(T_4) = \{T_7, T_8\}$ abgebrochen. Anschließend werden die Knoten und Kanten von T_3, T_4, T_5, T_7 und T_8 aus dem *wrf*-USG entfernt. □

Lokale Scheduler, in die ein CORCO integriert ist, sichern die CO- und RC-Eigenschaft eines Schedules. Wesentlich ist, daß der COCO bzw. CORCO nicht für das Scheduling der Datenzugriffsoperationen verantwortlich ist, sondern lediglich für die ordnungsgemäße Terminierung der Transaktionen. Sie können somit nur im Zusammenhang mit einem RAS oder einem kompletten Scheduler als vorgelagerter Komponente betrieben werden. Da die vorgestellten TTS-Komponenten *nichtblockierender* Natur sind, sichert eine Kombination mit nichtblockierenden RAS oder kompletten nichtblockierenden Schedulern verklemmungsfreie lokale Abarbeitungen. Der RAS kann mit beliebigen Scheduling-Strategien implementiert werden, ohne daß der CO-TTS die vom RAS realisierten Scheduleeigenschaften gefährdet. Dies gilt auch dann, wenn der TTS an einem atomaren "commit"-Protokoll teilnimmt.

In [Raz92] und [Rah94] wird unterstrichen, daß sich der Einsatz von lokalen, die CO-Eigenschaft realisierenden Schedulern vorteilhaft in föderierten Umgebungen auswirkt. Voraussetzung dabei ist jedoch, daß bei der Entscheidungsfindung über die Art der Terminierung globaler Transaktionen ein atomares "commit"-Protokoll wie das 2PC-Protokoll zum Einsatz kommt. Ist dies der Fall, so entstehen in den beteiligten Systemen bestimmte Entscheidungsintervalle, die den Zeitraum vom lokalen Votum für ein "commit" oder "abort" einer globalen Transaktion, der globalen Entscheidung und der lokalen Realisierung der globalen Terminierungsentscheidung umfassen. Können nun alle beteiligten Systeme gewährleisten, daß sich diese Intervalle nicht überlappen, so ist damit ein allgemeiner Ansatz für die Realisierung der globalen Serialisierbarkeit einer Abarbeitung gegeben.

Leider wird in dem in [Raz92] vorgestellten föderierten Ansatz die Möglichkeit der Ausführung lokaler Transaktionen nicht betrachtet. Es ist jedoch intuitiv klar, daß wenn alle lokalen Systeme ihre "commit"- Entscheidungen bzw. Entscheidungsintervalle entsprechend der lokalen Serialisierungsreihenfolge ordnen, sich die lokalen Intervalle nicht überlappen und ein AC-Protokoll realisiert wird, daß dann lokale Transaktionen keinen Einfluß auf die globale Serialisierbarkeit haben.

b) Die implizite Ticketmethode

Auch in [GRS94] wird eine Methode vorgestellt, die die Eigenschaft lokal streng rücksetzbarer Schedules ausnutzt. Die *implizite Ticketmethode* (ITM) sichert globale Serialisierbarkeit, jedoch ohne explizit Ticketoperationen durchführen zu müssen, wie es bei der OTM und der CTM der Fall gewesen ist. Die ITM ist anwendbar, wenn alle beteiligten Datenbanksysteme *analoge Ausführungs- und Serialisierungsreihenfolgen* (AESO – "analogous execution and serialization order") realisieren. Diese Eigenschaft wird in [GRS94, BGRS91] wie folgt definiert:

Definition 4.12 (Analoge Ausführungs- und Serialisierungsreihenfolge) Es sei S ein serialisierbarer Schedule. Die Transaktionen in S haben dann eine analoge Ausführungs- und Serialisierungsreihenfolge, wenn für jedes Transaktionspaar T_i und T_j mit $c_i \prec_S c_j$ in S gilt, daß T_i in S auch vor T_j serialisiert wird. □

Diese Definition ist sowohl für view- als auch konfliktserialisierbare Schedules anwendbar. In [BGRS91] wird bewiesen, daß StRC-Scheduler konfliktserialisierbare Ausführungen mit analoger Ausführungs- und Serialisierungsreihenfolge erzeugen (vergleiche Abbildung 4.5). Realisieren alle CDBMSe analoge Ausführungs- und Serialisierungsreihenfolgen, so ist für jede globale Subtransaktion durch die "commit"-Reihenfolge ein *implizites Ticket* gegeben. Die ITM muß somit nur noch die Reihenfolge der Abarbeitung der "commits" überwachen und erhält so die Serialisierungsreihenfolgen der Subtransaktionen.

Unter Voraussetzung der gleichen Systemumgebung wie bei der OTM oder der CTM und analoger Ausführungs- und Serialisierungsreihenfolgen in den lokalen Systemen läuft die Abarbeitung einer Menge globaler Transaktionen $\mathcal{G}$ mittels ITM wie folgt ab: Der GTM setzt zunächst ein Timeout-Intervall für jede globale Transaktion aus $\mathcal{G}$ und reicht die Subtransaktionen zur lokalen Abarbeitung an die CDBMSe weiter. Diese bearbeiten die Subtransaktionen selbständig bis diese den sichtbaren "prepared-to-commit"-Zustand erreicht haben. Ohne Beschränkung der Allgemeinheit nehmen wir an, daß $G_1, G_2, \ldots, G_k$ aus $\mathcal{G}$ den "prepared-to-commit"-Zustand erreichen, bevor deren Timeout-Intervalle abgelaufen sind. Wir nehmen weiterhin an, daß G_1 diesen Zustand vor G_2 erreicht, d.h. mindestens eine Subtransaktion von G_2 erreicht den "prepared"-Zustand nachdem alle Subtransaktionen von G_1 ihn erreicht haben, G_2 ist vor G_3 bereit zum "commit" und G_{k-1} vor G_k. Die ITM erlaubt dann allen Subtransaktionen von G_1, ihre "commit"-Operation vor den Subtransaktionen von G_2 auszuführen, die Subtransaktionen von G_2 enden vor denen von G_3 und die G_{k-1}-Subtransaktionen vor den G_k-Subtransaktionen. Hat eine Subtransaktion von G_i den "prepared-to-commit"-Zustand nicht vor dem Ablauf des Timeout-Intervalls erreicht, so bricht der GTM diese globale Transaktion ab und startet sie erneut.

Die implizite Ticketmethode gewährleistet somit globale Serialisierbarkeit, wenn die lokalen Synchronisationsmechanismen analoge Ausführungs- und Serialisierungsreihenfolgen garantieren, jede globale Transaktion aus maximal einer Subtransaktion pro CDBMS besteht und jede Subtransaktion einen sichtbaren "prepared-to-commit"-Zustand hat.

In [GRS94] werden außerdem nichtkaskadierende Ticketmethoden (ACA OTM, ACA CTM) vorgestellt, die die ACA-Eigenschaften lokaler Schedules ausnutzen können. Da beide Methoden lediglich Verfeinerungen der jeweils ursprünglichen Methoden sind, soll hier auf eine ausführliche Darstellung verzichtet werden.

4.2.4 Rigorosität in DBMSen

Die letzte in diesem Kapitel vorgestellte Klasse von Schedulern realisiert noch restriktivere Ausführungspläne und kann dadurch zu mehr Effizienz bei der Durchsetzung globaler Serialisierbarkeit führen. Die Eigenschaft lokaler Schedules, die dabei genutzt werden soll, ist die der *Rigorosität*, die in [BGS92] folgendermaßen definiert wird:

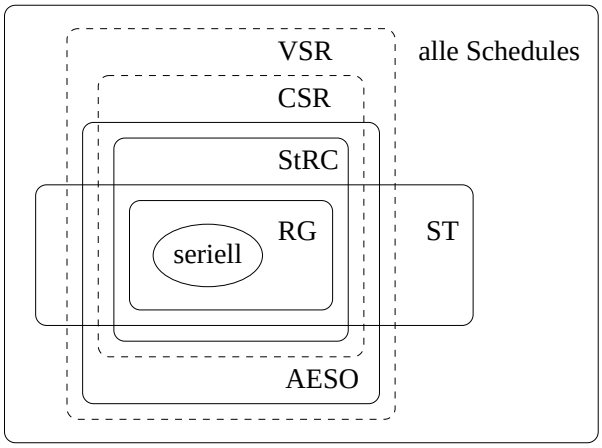

Abbildung 4.5: Beziehungen der vorgestellten Kriterien

Definition 4.13 (Rigorosität) Ein Schedule S_k ist *rigoros* (RG), wenn für alle Transaktionspaare T_i und T_j die folgende Bedingung gilt: Wenn T_i in S_k in einem direkten Konflikt zu T_j steht und T_j endet in S_k mit "commit" (c_j), dann wird die den Konflikt auslösende Operation von T_j nicht ausgeführt, bevor nicht c_i ausgeführt wurde. □

Die Beziehung der Rigorosität zu den Scheduleeigenschaften aus Abschnitt 3.1 und den vorangegangenen Kriterien stellt die Abbildung 4.5 graphisch dar. Die Rigorosität ist eine Verschärfung der Striktheit eines Schedules, insofern sie auch eine Verzögerung von Operationen verlangt, die einen rw-Konflikt auslösen, während die Striktheit dies nur für $ww-$ und wr-Konflikte forderte. Aus diesem Sachverhalt folgt, daß die Rigorosität auch eine Verschärfung der StRC-Eigenschaft ist, die ja lediglich für alle Konflikte eine "commit"-Sortierung entsprechend der Konfliktreihenfolge forderte. Folglich produzieren Scheduler, die die RG-Eigenschaft einer Abarbeitung sichern, genau wie die StRC-Scheduler konfliktserialisierbare Schedules mit analoger Ausführungs- und Serialisierungsreihenfolge.

Die Rigorosität lokaler Schedules wird beispielsweise durch das sehr populäre S2PL-Verfahren [BHG87] gesichert. Dieses Sperrverfahren hält in seiner ursprünglichen Definition sowohl alle Lese- als auch alle Schreibsperren einer Transaktion bis zu deren Abarbeitungsende ("abort" oder "commit"). Da dadurch Operationen anderer Transaktionen, die einen Konflikt zur sperrenden Transaktion erzeugen würden, in ihrer Ausführung bis zur Beendigung der sperrenden Transaktion verzögert werden, sichert das S2PL-Protokoll rigorose Abarbeitungen. Leider werden in der Literatur mit der Bezeichnung "S2PL-Scheduler" häufig unterschiedliche Implementierungen für das Sperrverfahren und damit unterschiedliche Scheduleeigenschaften verbunden. So versteht [Raz92] unter einem S2PL-Scheduler einen sperrenden Transaktionsverwaltungsmechanismus, der lediglich die "Striktheit" von Abarbeitungen sichert und dafür nur die Schreibsperren bis zum Transaktionsende hält. Ein Scheduler, der auch die Lesesperren erst zum Transaktionsende freigibt, wird in [Raz92] als "S-S2PL-Scheduler" ("strong strict two phase locking")

bezeichnet. In [BGRS91] wird deshalb die Bezeichnung "rigoroser 2PL-Scheduler" (R2PL – "rigorous two phase locking") für Scheduler vorgeschlagen, die durch das Halten der Schreib- und Lesesperren bis an das Transaktionsende die Rigorosität lokaler Schedules sicherstellen. In diesem Artikel wird außerdem erläutert, wie weitere klassische Synchronisationsverfahren, z.B. TO-Scheduler oder Serialisierbarkeitsgraphentester, modifiziert werden können, um rigorose Ausführungspläne zu realisieren.

Betrachtet man die im Beispiel 3.2 gegebenen Abarbeitungsfolgen unter dem Einsatz lokaler R2PL-Scheduler, ergeben sich folgende lokale Situationen (die Operationen in den eckigen Klammern stellen die noch zu bearbeitenden Schritte nach Beispiel 3.2 dar):

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

Im Knoten s_2 hält LT_4 die Sperre für c bis zum "commit". Folglich ist es GT_1 nicht möglich, auf c zuzugreifen, und GT_1 wird in s_2 bis zur Beendigung von LT_4 verzögert. Bleibt die Ausführungsreihenfolge von GT_1 und GT_2 erhalten, wir sind in Beispiel 3.2 von seriellen Ausführungen der globalen Transaktionen ausgegangen, so wird die ungewünschte Serialisierungsreihenfolge $GT_2 \rightarrow LT_4 \rightarrow GT_1$ vermieden.

Die Rigorosität allein reicht jedoch nicht zur Sicherung globaler Serialisierbarkeit aus, da auch lokal rigorose Ausführungen globaler Transaktionen denkbar sind, die unterschiedliche Serialisierungsreihenfolgen globaler Transaktionen realisieren, wie [BGS92] beispielhaft zeigen:

$$S_1 : w_1(a) c_1 r_3(a) w_3(b) c_3 r_2(b) c_2$$
$$S_2 : w_2(c) c_2 r_4(c) w_4(d) c_4 r_1(d) c_1$$

Jeder lokale Schedule ist rigoros (sogar seriell), jedoch werden die globalen Transaktionen lokal unterschiedlich serialisiert ($s_1 : GT_1 \rightarrow LT_3 \rightarrow GT_2$ bzw. $s_2 : GT_2 \rightarrow LT_4 \rightarrow GT_1$) und so die globale Serialisierbarkeit verletzt.

Sichert jedoch der GTM, daß die "commit"-Operation einer globalen Transaktion erst zur Abarbeitung eingereicht wird, wenn alle vorangegangenen Datenbankoperationen der globalen Transaktion ihre Arbeit in den lokalen Systemen beendet haben, dann ist die Rigorosität lokaler Abarbeitungen zur Gewährleistung der globalen Serialisierbarkeit ausreichend. Globale Transaktionen, die diese Bedingung erfüllen, werden als *"commitverzögerte"* Transaktionen bezeichnet. Der GTM kann somit in derartigen Umgebungen durch die Kontrolle der "commit"-Reihenfolge der globalen Transaktionen in den teilnehmenden Knoten einer Föderation die globale Serialisierbarkeit gewährleisten. In [BGRS91] wird folgender Satz bewiesen:

Satz 4.4 Sind in einem föderierten Datenbanksystem alle globalen Transaktionen "commitverzögert" und realisieren alle CDBMSe rigorose Ausführungspläne, so ist dadurch die Konsistenz der globalen Transaktionsausführung gesichert. □

Diese Aussage gestattet entscheidende Vereinfachungen beim globalen Transaktionsmanagement, wenn alle beteiligten DBMSe rigoroser Natur sind, da die globale Konsistenzerhaltung und Serialisierbarkeit allein durch die Steuerung der "commit"-Reihenfolgen der globalen Transaktionen in den lokalen Systemen zu realisieren sind.

So kann beispielsweise der Knotengraphenalgorithmus der ADDS-Transaktionsverwaltung aus Abschnitt 4.2.1 für rigorose CDBMSe in einer vereinfachten Form angewendet werden. Zunächst wird durch die lokale Rigorosität das Problem des Löschens von globalen Transaktionen aus dem Graphen dahingehend gelöst, daß globale Transaktionsknoten schon zum "commit"-Zeitpunkt aus dem Graphen entfernt werden können, da überlappende Bearbeitungen nicht konfliktbehaftet sind. Die Aufgabe des Knotengraphen bestand darin, in den einzelnen Datenbankknoten eine überlappende Abarbeitung von Subtransaktionen, die miteinander in einem Konflikt standen, zu verhindern. Da dies nun bereits durch die Rigorosität der lokalen Schedules gewährleistet wird, ist vielmehr der Einsatz eines Knotengraphen gänzlich überflüssig. Subtransaktionen können jederzeit an die lokalen Systeme weitergereicht werden. Das ADDS-Transaktionsverwaltungsprotokoll sendet eine globale "commit"-Operation erst zu den beteiligten Knoten, wenn alle Datenbankoperationen der Subtransaktionen vollständig ausgeführt wurden.

Die implizite Ticketmethode aus dem vorangegangenen Abschnitt ist ebenso für Föderationen mit lokal rigorosen DBMSen anwendbar, da Rigorosität analoge Ausführungs- und Serialisierungsreihenfolgen impliziert. Vielmehr ist durch die Rigorosität lokaler Abarbeitungen ein in der Praxis bedeutend besser umsetzbares Kriterium gegeben, als durch die Forderung nach analogen Ausführungs- und Serialisierungsreihenfolgen. Gleiches gilt für weitere Verfahren und Ansätze, so daß [BGRS91] zu dem Schluß kommen, daß das altruistische Sperrverfahren, die Knotengraphenmethode, die ITM und die 2PC-Agent-Methode (vergleiche Abschnitt 6.2) globale Serialisierbarkeit garantieren, wenn alle CDBMSe rigorose Scheduler besitzen und jede globale Transaktion maximal eine Subtransaktion pro Datenbankknoten hat. Unter diesen Voraussetzungen und der Nutzung eines AC-Protokolls ist es somit möglich, einen allgemeinen Top-Down-Ansatz für ein globales Transaktionsmanagement zu implementieren, das globale Serialisierbarkeit gewährleistet, ohne Veränderungen an den lokalen DBMSen zu erfordern.

Kapitel 5

Alternative Korrektheitskriterien

Im vorangegangenen Kapitel sind einige Kriterien und Methoden zur Gewährleistung globaler Serialisierbarkeit vorgestellt wurden. Es hat sich gezeigt, daß die Sicherung global serialisierbarer Abarbeitungen in föderierten Datenbanksystemen nur sehr schwer zu realisieren ist. Restriktionen an globale Abarbeitungen und lokale Scheduleeigenschaften waren nötig, um die Forderung nach konsistenzerhaltenden globalen Transaktionsabarbeitungen zu erfüllen. Besonders die Forderung nach lokaler Autonomie konnte damit nur stark eingeschränkt erfüllt werden. Außerdem erscheint die Leistungsfähigkeit dieser Realisierungsansätze in bestimmten Umgebungen eher gering, da entweder eine geringe Parallelität der globalen Transaktionen oder häufige Transaktionsabbrüche in Kauf genommen werden, um die globale Serialisierbarkeit zu sichern. Die Auswirkungen möglicher Fehler während der Abarbeitung, die die Problematik weiter verschärfen würden, sind dabei noch gar nicht in Betracht gezogen wurden.

Aus den angeführten Gründen werden in verschiedenen Arbeiten zu dieser Thematik alternative Korrektheitskriterien zur globalen Serialisierbarkeit vorgeschlagen, die in der Regel eine Abschwächung der ursprünglichen Konsistenzforderungen vornehmen. Einige dieser alternativen Ansätze sollen innerhalb dieses Kapitels kurz vorgestellt werden. Wie bereits in Kapitel 4 soll dabei zur inhaltlichen Darstellung zunächst von fehler- und abbruchfreien Umgebungen ausgegangen werden.

5.1 Lokale Serialisierbarkeit

Im Kapitel 4 war unter der Annahme konsistenzerhaltender globaler Transaktionen die Sicherung der globalen Serialisierbarkeit ein Garant dafür, daß auch die jeweilig realisierten Ablaufpläne die Datenbank von einem konsistenten in einen konsistenten Zustand überführen. Da in diesem Kapitel weniger restriktive Forderungen an die Abarbeitungen gestellt werden sollen, müssen andere Wege gefunden werden, die Erhaltung der Datenbankkonsistenz durch die Abarbeitung globaler Transaktionen zu sichern.

Die Konsistenz einer Datenbank wird in der Regel durch Integritätsbedingungen definiert, wobei man in föderierten Systeme lokale und globale Bedingungen unterscheiden

kann. Lokale Integritätsbedingungen beziehen sich nur auf Datenobjekte des entsprechenden Datenbanksystems, wogegen globale Integritätsbedingungen für Datenobjekte definiert werden, die über mehrere Datenbankknoten verteilt sind. Sowohl globale als auch lokale Integritätsbedingungen können statischer oder dynamischer Natur sein. Statische Integritätsbedingungen definieren Bedingungen für die Korrektheit bestimmter Datenbankzustände, wohingegen durch dynamische Bedingungen die Korrektheit von Zustandsübergängen spezifiziert wird. In diesem Kapitel sollen jedoch nicht die Mechanismen zur expliziten Realisierung von Integritätsbedingungen untersucht werden, sondern es wird wie im vorangegangenen Kapitel davon ausgegangen, daß alle Integritätsbedingungen durch die Transaktionen abgesichert sind. Es wird lediglich untersucht, inwieweit auch nach einer Lockerung der globalen Serialisierbarkeitsforderung die Sicherung der Integritätsbedingungen garantiert werden kann.

In diesem Abschnitt soll nur von lokalen Integritätsbedingungen ausgegangen werden. Einige Publikationen argumentieren, daß sich in föderierten Systemen die CDBMSe unabhängig entwickelt haben und unabhängig bleiben sollen und man deshalb auf globale Bedingungen verzichten sollte. Diese Ansicht ist jedoch besonders in eng gekoppelten Systemen sehr umstritten. Für lose gekoppelte Föderationen scheint diese Herangehensweise jedoch geeignet, da aufgrund der Vielzahl föderierter Schemata in diesen Umgebungen eine einheitliche Umsetzung globaler Integritätsbedingungen kaum sinnvoll möglich ist. Die lokalen Bedingungen sind leicht zu erhalten, da jeder Knoten mit lokalen Synchronisationsmechanismen ausgestattet ist, die die Serialisierbarkeit der lokalen Schedules gewährleisten. Das bedeutet, daß, auch wenn die globale Serialisierbarkeit verletzt wird, zumindest die lokalen Integritätsbedingungen durch die lokale Serialisierbarkeit bewahrt bleiben. Folgende Definitionen aus [BGS92] formalisieren diesen Sachverhalt:

Definition 5.1 (Lokale Serialisierbarkeit) Ein globaler Schedule S heißt *lokal serialisierbar* (LSR – "local serializable"), wenn für jeden Knoten s_i $(1 \leq i \leq n)$ der lokale Schedule serialisierbar ist. □

Die Ausführung einer Menge von Transaktionen $\mathcal{T}$ überführt einen gegebenen Anfangszustand der Datenbank in einen bestimmten Endzustand. Eine solche Ausführung wird dabei durch einen Schedule dargestellt, der anhand einer Folge von "read-", "write-", "abort-" und "commit"-Operationen den Datenbankzustand spezifiziert, den jede Transaktion $T_k \in \mathcal{T}$ in einer Ausführung liest.

Definition 5.2 (Strenge Korrektheit) Eine Ausführung ist *streng korrekt*, wenn der erreichte Endzustand konsistent bezüglich der lokalen Integritätsbedingungen ist und jeder Zwischenzustand, den eine Transaktion $T_k \in \mathcal{T}$ gelesen hat, ebenfalls konsistent war. □

[VG93] beschreiben die strenge Korrektheit in einer leicht abgewandelten Form. Sie unterteilen die in einer Föderation zu verwaltenden Daten in lokale und globale Objekte und unterteilen die Bedingungen entsprechend der involvierten Daten in lokale, globale

und gemischte Integritätsbedingungen (dieser Ansatz soll auch im nächsten Abschnitt zur zweistufigen Serialisierbarkeit herangezogen werden). Eine Ausführung ist demnach dann streng korrekt, wenn ausgehend von einem konsistenten Anfangszustand ein konsistenter Endzustand erreicht wird und alle von den einzelnen Transaktionen gelesenen Zwischenzustände konsistent sind, wobei alle drei Arten von Integritätsbedingungen zu berücksichtigen sind.

Hier soll jedoch weiter die Begriffsbestimmung aus [BGS92] benutzt werden, die sich auf die Wahrung der lokalen Integritätsbedingungen beschränkt. Um zu zeigen, daß die LSR strenge Korrektheit gewährleistet, müssen bestimmte "ungewöhnliche" Transaktionen ausgeschlossen werden, wie folgendes Beispiel illustriert:

Beispiel 5.1 Ein FDBS bestehe aus zwei Knoten: s_1 mit dem Datenobjekt a und s_2 mit dem Datenobjekt b. Es existieren zwei lokale Integritätsbedingungen: $a > 0$ auf s_1 und $b > 0$ auf s_2. Wir betrachten die folgenden globalen Transaktionen:

$$\begin{array}{ll} GT_1: & a = -1 \\ & if\ (b > 0)\ then\ a = 1 \\ GT_2: & b = -1 \\ & if\ (a > 0)\ then\ b = 1 \end{array}$$

Beide Transaktionen sind möglich, da sie einen konsistenten Ausgangszustand in einen konsitenten Endzustand überführen. Jedoch kann es unter bestimmten Umständen zu einer Verletzung der Integritätsbedingungen kommen, wie folgende Ausführungen zeigen:

$$\begin{array}{l} S_1 : w_1(a = -1) r_2(a = -1) \\ S_2 : w_2(b = -1) r_1(b = -1) \end{array}$$

Obwohl jeder Schedule serialisierbar ist, sind die durch sie erreichten Endzustände inkonsistent. □

Eine einfache Möglichkeit, das obige Problem zu beheben, ist die Anwendung eines lokalen 2PL-Schedulers. Da die schreibende Transaktion eines Knotens die gehaltene Schreibsperre frühestens nach der Leseoperation auf dem anderen Knoten abgeben kann (da in Abhängigkeit vom Ergebnis der Leseoperation eventuell noch mal geschrieben werden muß), würden die Schedules zwar zu einer Verklemmung führen, die Inkonsistenz der Datenbank aber vermeiden.

Andere Möglichkeiten, das Problem aus Beispiel 5.1 zu vermeiden, bilden Restriktionen an die Struktur der verschiedenen Transaktionen:

- *Forcierung datenbankerhaltender Transaktionen*

  Das Problem kann vermieden werden, wenn alle Transaktionen eines Knotens die dortige Konsistenz erhalten, ungeachtet der Zustände in anderen Knoten. Erfüllen Transaktionen diese Bedingung, so sind sie *lokal datenbankerhaltend* (LDP - "local database preserving"). Sind alle Transaktionen LDP und alle Integritätsbedingungen lokal, garantieren LSR-Schedules strenge Korrektheit. Im Beispiel 5.1 sind die Transaktionen nicht LDP, da beispielsweise aus Sicht des Knotens s_1 die Transaktion GT_1 die Datenbank von einem lokal konsistenten Zustand ($a > 0$) in einen lokal inkonsistenten Zustand ($a = -1$) überführt.

- *Forcierung von feststehenden Transaktionsstrukturen*

  Transaktionen, die immer aus den gleichen r/w-Operationsmustern bestehen, heißen Transaktionen mit *feststehender Struktur*. Ungeachtet der Werte der gelesenen Datenobjekte bestehen solche Transaktionen immer aus den gleichen Operationen in einer konstanten Reihenfolge. Haben alle Transaktionen eine feste Struktur und sind alle Integritätsbedingungen lokal, garantieren LSR-Schedules strenge Korrektheit. Die Transaktionen im Beispiel 5.1 haben keine feste Struktur. GT_1 mit feststehender Struktur könnte wie folgt aussehen:

$$\begin{aligned} GT_1: \quad & a = -1 \\ & if\ (b > 0)\ then\ a = 1\ else\ a = -1 \end{aligned}$$

  Wird GT_2 ebenfalls entsprechend modifiziert, kann das obige Problem nicht mehr auftreten, da dadurch die lokale Serialisierbarkeit verletzt würde.

Die Forderung nach ausschließlicher Abarbeitung von LDP-Transaktionen erscheint praktikabel. Lokale Transaktionen sind immer LDP, ebenso die meisten globalen Subtransaktionen. Sind letztere einmal nicht lokal datenbankerhaltend, kann man sie recht leicht LDP machen, vorausgesetzt, daß die lokalen Bedingungen dem GTM bekannt sind. Dagegen erscheint die Forderung nach feststehenden Strukturen weniger sinnvoll, da im Falle von lokalen Transaktionen dadurch die lokale Autonomie verletzt wird.

In [DE89] wird eine weitere Strategie vorgeschlagen, um LSR-Schedules bedingungserhaltend zu machen. Eine Transaktion T_i hat keine Werteabhängigkeiten (die Transaktion ist NVD – *"no value dependencies"*), wenn ihre Aktionen in einem Knoten in keiner Weise von in anderen Knoten gelesenen Werten abhängen. Sowohl GT_1 als auch GT_2 aus Beispiel 5.1 haben Werteabhängigkeiten. Die Menge der NVD-Transaktionen ist echte Untermenge der LDP-Transaktionen. Die Forcierung von NVD-Transaktionen sichert die strenge Korrektheit von Ausführungen, ist aber nach [DE89] restriktiver als nötig.

5.2 Zweistufige Serialisierbarkeit

Die Idee der *zweistufigen Serialisierbarkeit* beruht auf einer Erweiterung des vorangegangenen Ansatzes. Man unterteilt dabei in jedem Knoten die zu verwaltenden Daten in eine Menge *lokaler* und eine Menge *globaler* Datenobjekte. Entsprechend der Typen der involvierten Datenobjekte kann man drei Arten von Integritätsbedingungen unterscheiden [VG93, BGS92]:

- *Lokale Bedingungen* betreffen ausschließlich die lokalen Datenobjekte eines einzelnen Datenbankknotens und werden auch von diesem verwaltet.
- *Globale Bedingungen* werden ausschließlich über globale Datenobjekte definiert, die über verschiedene Knoten verteilt sein können.
- *Gemischte Bedingungen* werden über lokale und globale Datenobjekte definiert, die sich jedoch auf einem einzigen Knoten befinden müssen.

Die Haupteinschränkung dieses Modells ist, daß lokale Transaktionen keine globalen Daten modifizieren dürfen, d.h. lokale Transaktionen dürfen globale Daten nur lesen, wogegen globale Transaktionen jede Art von Daten lesen und schreiben dürfen. Für FDBSe ist diese Einschränkung durchaus praktikabel, die Originaldatenbänke bilden die lokalen Daten und können weiterhin uneingeschränkt durch die ursprünglichen Transaktionen gelesen und modifiziert werden. Durch die Teilnahme an einer Föderation kommen neue globale Datenobjekte hinzu, die in den einzelnen Knoten gespeichert werden und ausschließlich durch neue globale Transaktionen modifiziert werden können. Neue lokale Transaktionen haben das Recht, die globalen Daten zu lesen, können diese aber nicht modifizieren, da diese Daten in globale Integritätsbedingungen involviert sein können.

Lokale und gemischte Integritätsbedingungen sollten weder direkt noch indirekt von Datenobjekten anderer Knoten abhängen. Beispielsweise besteht eine Föderation aus zwei Knoten: s_1 mit dem lokalen Datenobjekt a und dem globalen Datenobjekt b und s_2 mit dem globalen Datenobjekt c und es existiert die globale Integritätsbedingung $b = c$. Eine gemischte Bedingung $a = b$ ist in diesem Fall nicht erlaubt, weil dies die Abhängigkeit $a = c$ eines lokalen Datenobjekts von Datenobjekten entfernter Knoten induziert.

Der GTM kontrolliert die Abarbeitung der globalen Transaktionen und sichert deren Serialisierbarkeit im globalen Schedule. Gleichzeitig sichern lokale Synchronisationsmechanismen die lokale Serialisierbarkeit. Daraus ergibt sich eine zweistufige Serialisierbarkeit, die in [VG93] wie folgt definiert wird:

Definition 5.3 (Zweistufige Serialisierbarkeit) Ein globaler Schedule S ist *zweistufig serialisierbar* (2LSR - "two level serializable"), wenn alle lokalen Projektionen serialisierbar sind, d.h. S ist LSR, und die Projektion auf die Menge der in S vorkommenden globalen Transaktionen serialisierbar ist. □

Global serialisierbare Schedules sind immer 2LSR, die Umkehrung dieser Aussage gilt jedoch nicht, wie man sich leicht am folgenden Beispiel klarmachen kann [BGS92, VG93]:

Beispiel 5.2 Ein FDBS bestehe aus zwei Knoten: s_1 mit dem lokalen Datenobjekt a und den globalen Datenobjekten b und c, s_2 mit dem globalen Datenobjekt d. Weiterhin seien eine globale $(d > 0 \rightarrow (b > 0 \vee c > 0))$ und eine gemischte $(a > 0 \rightarrow b > 0)$ Integritätsbedingung gegeben. Wir betrachten eine lokale und zwei globale Transaktionen:

$$\begin{array}{ll} GT_1: & if\ (a <= 0)\ then\ c = 1\ else\ c = -1 \\ & d = 1 \\ GT_2: & if\ (a <= 0)\ then\ b = -1\ else\ b = 1 \\ & d = -1 \\ LT_3 & a = -1 \end{array}$$

Als Anfangzustand wird angenommen, daß alle Datenobjekte den Wert "1" haben und dann folgende Schedules ausgeführt werden:

$$S_1 : r_1(a = 1)w_1(c = -1)w_3(a = -1)r_2(a = -1)w_2(b = -1)$$
$$S_2 : w_2(d = -1)w_1(d = 1)$$

Beide Schedules sind offensichtlich lokal serialisierbar ist, jedoch sind die durch sie erreichten Endzustände inkonsistent ($d > 0$ aber $(b < 0 \wedge c < 0)$). Obwohl der globale Schedule 2LSR ist, ist er nicht global serialisierbar. Im Knoten s_1 gilt die Serialisierungsreihenfolge $GT_1 \rightarrow LT_3 \rightarrow GT_2$, im Knoten s_2 dagegen $GT_2 \rightarrow GT_1$. □

Aus Sicht der einzelnen Knoten sind die Transaktionen LDP und alle lokalen und gemischten Integritätsbedingungen werden garantiert. Jedoch wird die globale Integritätsbedingung verletzt, da aus globaler Sicht die Transaktion GT_1 nicht gültig ist, da sie nur einen konsistenten Zustand erreicht, wenn die gemischte Bedingung erfüllt ist. Eine Transaktion heißt *global datenbankerhaltend* (GDP - "global database preserving"), wenn sie alle globalen Integritätsbedingungen erfüllt, ungeachtet der Werte lokaler Datenobjekte. [BGS92, VG93] kommen zu dem Schluß, daß wenn alle Transaktionen LDP und GDP sind, jeder zweistufig serialisierbare Schedule strenge Korrektheit garantiert.

Erfüllen die Transaktionen bestimmte Zugriffsmuster, lassen sich die LDP/GDP-Anforderungen lockern. Wird globalen Transaktionen verboten, auf lokale Daten zuzugreifen, kann die GDP-Forderung fallengelassen werden. Wird zusätzlich davon ausgegangen, daß lokale Transaktionen globale Daten nicht lesen können und daß globale Transaktionen keine lokalen Daten schreiben, d.h. es wird von einer völligen Entkopplung lokaler und globaler Daten ausgegangen, dann sind 2LSR-Schedules ohne zusätzliche Restriktionen immer global serialisierbar.

Gibt es keine gemischten Bedingungen kann man die GDP-Forderung ebenfalls fallenlassen, wie [BGS92] zeigen. Weiterhin kann man in diesem Fall die LDP-Forderungen

unter bestimmten Voraussetzungen abschwächen, was hier jedoch nicht weiter erläutert werden soll.

In [OAB94] wird ein Top-Down-Ansatz zur Realisierung von globalen 2LSR-Schedules vorgestellt. Da dieser Ansatz in vielen Punkten synchron zum Top-Down-Ansatz aus [DE90] läuft, soll auf eine ausführliche Darstellung verzichtet und bloß die wesentlichen Merkmale und Unterschiede genannt werden. Wie beim [DE90]-Ansatz zur Realisierung global serialisierbarer Ausführung, bestimmt der GTM zunächst eine Serialisierungsreihenfolge der globalen Transaktionen O und gibt diese zusammen mit den Subtransaktionen zur lokalen Umsetzung an die Server weiter. Im Gegensatz zu [DE90] wird die Bandbreite der dabei handhabbaren lokalen Scheduler von [OAB94] bedeutend weiter gefaßt und damit die lokale Autonomie nicht durch die Forderung nach SP-basierten lokalen Serialisierungsmechanismen beschränkt. Realisieren beispielsweise die lokalen Scheduler lediglich einfache serielle Ausführungen ("unlabeled black boxes"), kann der Server durch Ticketoperationen Konflikte zwischen globalen Subtransaktionen forcieren und O dadurch umsetzen, daß eine Subtransaktion erst zur Abarbeitung eingereicht wird, wenn die in O vorangehenden Subtransaktionen ihre Ticketoperationen ausgeführt haben. SP-basierte lokale Scheduler werden von den Servern genauso behandelt, wie es in [DE90] vorgeschlagen wurde. Sichern die lokalen Scheduler die Rigorosität der Ausführung, wird eine Subtransaktion erst dann zur Ausführung gebracht, wenn die in O vorangehende globale Subtransaktion beendet wurde.

Der wesentliche Unterschied zu [DE90] besteht darin, daß bei der zweistufigen Serialisierbarkeit die indirekten Konflikte nicht betrachtet werden müssen. Das bedeutet, daß ein Server die Reihenfolge O zwischen zwei globalen Subtransaktionen nur dann durchsetzen muß, wenn diese beiden Transaktionen in dem entsprechenden Knoten in einem direkten Konflikt zueinanderstehen. Ist das nicht der Fall, kann der Server eine beliebige Ausführungsreihenfolge wählen, ohne die zweistufige Serialisierbarkeit zu verletzen. Zur Umsetzung dieses wesentlichen Aspekts der zweistufigen Serialisierbarkeit wählen [OAB94] einen graphenbasierten Ansatz und definieren einen OCDAG (*"oriented conflict data access graph"*) für jeden Knoten des föderierten Datenbanksystems. Der OCDAG_k des Knotens s_k ist ein gerichteter Graph, dessen Knotenmenge die Menge der in diesem Datenbankknoten abzuarbeitenden globalen Subtransaktionen umfaßt und der genau dann eine gerichtete Kante $GST_i \rightarrow GST_j$ zwischen zwei globalen Subtransaktionen enthält, wenn GT_i in O vor GT_j steht und GST_i in s_k zu GST_j in einem direkten Konflikt steht. Der Server muß damit lediglich für diejenigen Subtransaktionen die Ausführungsbedingungen beachten, die zumindest eine eingehende Kante im entprechenden OCDAG haben. Ein entsprechender Algorithmus wird in [OAB94] vorgestellt. Der hier beschriebene Ansatz realisiert durch die Lockerung der globalen Serialisierbarkeitsforderung einen größeren Parallelitätsgrad als der im Abschnitt 4.2.2 vorgestellte Ansatz und gewährt den CDBMSen gleichzeitig mehr lokale Autonomie.

Ein Vorläufer des 2LSR-Ansatzes war die Quasiserialisierkeit von [DE89], die im folgenden Abschnitt erläutert werden soll.

5.3 Quasiserialisierbarkeit

Die *Quasiserialisierbarkeit* (QSR) ist ein Korrektheitskriterium, das von [DE89] im Zuge des InterBase-Projekts des Computer Sciences Department der Purdue Universität vorgestellt wurde. Dieses Projekt beschäftigte sich mit der Entwicklung eines FDBMSs, das atomare "Updates" über verschiedene Datenbanken gestattet. Die Quasiserialisierbarkeit ist eine Erweiterung der globalen Serialisierbarkeit, da aufgrund der hierachischen Struktur globaler Synchronisationsmechanismen und der Autonomie der lokalen Datenbanksysteme die Sicherung globaler Serialisierbarkeit zu den einleitend erwähnten Problemen führt.

[DE89] gehen dabei von dem Abschnitt 3.2 vorgestellten Transaktionsmodell aus, nehmen jedoch für die Formulierung ihres Korrektheitskriteriums gewisse Einschränkungen vor. So werden keine globalen Integritätsbedingungen unterstützt, da dies der Forderung nach lokaler Autonomie der beteiligten Datenbanksysteme widerspricht. Ebenso wird davon ausgegangen, daß die Subtransaktionen einer globalen Transaktion NVD sind. Diese Annahme ist zwar nicht notwendig für die QSR, erleichtert jedoch die nachfolgende Darstellung des Ansatzes.

Die Grundidee dieses Ansatzes besteht darin, daß zur Erhaltung der globalen Datenbankkonsistenz globale Transaktionen in einer serialisierbaren Ausführung unter besonderer Berücksichtigung der Effekte lokaler Transaktionen ablaufen. Grundlage bildet die Definition eines *quasiseriellen Schedules*, der im Gegensatz zum seriellen Schedule nur erfordert, daß globale Transaktionen seriell ablaufen. Zusammen mit der Serialisierbarkeit lokaler Ausführungen ist dies ausreichend, um die Korrektheit globaler Synchronisationsmechanismen in föderierten Umgebungen zu gewährleisten:

Definition 5.4 (Quasiserieller Schedule) Ein globaler Schedule S, bestehend aus einer Menge lokaler Schedules S_i $(1 \leq i \leq n)$, ist *quasiseriell*, wenn alle lokalen Schedules (konflikt-) serialisierbar sind und wenn es eine totale Reihenfolge O aller globalen Transaktionen derart gibt, daß für jedes Transaktionspaar GT_k und GT_l mit GT_k vor GT_l in O gilt, daß in allen Knoten, in denen beide Transaktionen ausgeführt werden, die Operationen von GT_k vollständig vor den Operationen von GT_l ausgeführt werden. □

Definition 5.5 (Quasiserialisierbarer Schedule) Ein Schedule ist *quasiserialisierbar* (QSR), wenn er (konflikt-) äquivalent zu einem quasiseriellen Schedule ist. □

Demnach sind in einem QSR-Schedule alle lokalen Schedules serialisierbar. Außerdem werden die globalen Transaktionen in einer serialisierbaren Form abgearbeitet, wobei sowohl direkte als auch indirekte Konflikte betrachtet werden[1]. Das folgende Beispiel illustriert diesen Ansatz ([DE89, VG93]):

[1] Implizit wird in [DE89] davon ausgegangen, daß die indirekten Konflikte globaler Transaktionen in der Reihenfolge O bereits berücksichtigt sind.

Beispiel 5.3 Ein FDBS bestehe aus zwei Knoten: s_1 mit den Datenobjekten a und b, s_2 mit den Datenobjekten c, d und e. GT_1 und GT_2 seien zwei globale Transaktionen:

$$GT_1 : w_1(a)r_1(d)$$
$$GT_2 : r_2(b)r_2(c)w_2(e)$$

Zusätzlich existieren zwei lokale Transaktionen LT_3 und LT_4 auf den Knoten s_1 und s_2:

$$LT_3 : r_3(a)w_3(b)$$
$$LT_4 : w_4(d)r_4(e)$$

Folgende lokale Schedules werden in s_1 bzw. in s_2 ausgeführt:

$$S_1 : w_1(a)c_1r_3(a)w_3(b)c_3r_2(b)c_2$$
$$S_2 : r_2(c)w_4(d)r_1(d)c_1w_2(e)c_2r_4(e)c_4$$

Beide Schedules sind offensichtlich serialisierbar. Da S_1 seriell ist, ist S_1 auch quasiseriell. S_2 ist konfliktäquivalent zu folgendem quasiseriellen Schedule:

$$S_2' : w_4(d)r_1(d)c_1r_2(c)w_2(e)c_2r_4(e)c_4$$

Demnach ist ein globaler Schedule, der aus den lokalen Schedules S_1 und S_2 besteht, quasiserialisierbar. Die globale Serialisierbarkeit von S ist dagegen verletzt, da in den Knoten unterschiedliche Serialisierungsreihenfolgen erzeugt werden ($s_1 : GT_1 \rightarrow LT_3 \rightarrow GT_2$, aber $s_2 : GT_2 \rightarrow LT_4 \rightarrow GT_1$). □

[DE89] stellen einen Quasiserialisierbarkeitsgraphen QSG(S) vor, der die globalen Transaktionen als Knotenmenge hat und eine gerichtete Kante zwischen den Knoten zweier globaler Transaktionen enthält, wenn diese miteinander in einem Konflikt stehen ($GT_i \rightarrow GT_j$). Es wird in [DE89] bewiesen, daß ein globaler Schedule S genau dann quasiserialisierbar ist, wenn alle lokalen Schedules S_i ($1 \leq i \leq n$) (konflikt-) serialisierbar sind und QSR(S) ist azyklisch. Im Beispiel 5.3 ist QSR(S) im Gegensatz zum globalen Serialisierbarkeitsgraphen azyklisch.

Zur Überprüfung der Korrektheit des QSR-Ansatzes werden die Auswirkungen quasiserialisierbarer Abarbeitungen auf die unterschiedlichen Integritätsbedingungen untersucht. Wie eingangs bereits erwähnt, wird davon ausgegangen, daß keine globalen Integritätsbedingungen existieren, da dies der lokalen Autonomie widerspricht. In [DE89] wird gezeigt, daß durch die Serialisierbarkeit auf unterschiedlichen Ebenen alle anderen Integritätsbedingungen erfüllt werden und somit die QSR ein mögliches Korrektheitskriterium in FDBSen darstellt.

[BGS92] stellen fest, daß quasiserialisierbare Ausführungen streng korrekt sind, wenn es keine gemischten Integritätsbedingungen gibt. Da die QSR-Schedules eine echte Untermenge der 2LSR-Schedules bilden[2], ist die letzte Aussage ein Spezialfall der allgemeineren aus dem Abschnitt 5.2.

[2]Beispiel 5.2 zeigt einen 2LSR-Schedule, der nicht quasiserialisierbar ist.

Ein Umsetzungsansatz für einen möglichen GTM, der Quasiserialisierbarkeit realisiert, wird in [DE90] dargestellt. Der dort vorgestellte Top-Down-Ansatz zur Realisierung von O-Serialisierbarkeit mit Hilfe sogenannter "Stub-Prozesse" ist ebenso geeignet, O-Quasiserialisierbarkeit zu gewährleisten, wenn die gestellten Anforderungen entsprechend gelockert werden. Konkret ist es im QSR-Ansatz ausreichend, daß alle Scheduler die einfache Serialisierbarkeit lokaler Abarbeitungen garantieren. Die Quasiserialisierbarkeit wird dann dadurch erreicht, daß ein "Stub-Prozeß" eine globale Subtransaktion GST_i erst dann zur Abarbeitung an das CDBMS weiterreicht, wenn die Abarbeitung aller globalen Subtransaktionen GST_j, die in O der Subtransaktion GST_i vorangehen, im entsprechenden Datenbankknoten bereits beendet ist.

5.4 Weitere alternative Korrektheitskriterien

Die vorangegangenen Ansätze haben gezeigt, wie globale und lokale Integritätsbedingungen in föderierten Datenbanksystemen durch die Realisierung serialisierbarer Schedules auf unterschiedlichen Ebenen durchgesetzt werden können, und gaben damit alternative Korrektheitskriterien zur globalen Serialisierbarkeit in föderierten Umgebungen. So sicherte die LSR-Eigenschaft globaler Schedules die Realisierung aller lokalen und gemischten Integritätsbedingungen, sofern die einzelnen Transaktionen dies tun, und die 2LSR-Eigenschaft sicherte zusätzlich die Durchsetzung der globalen Integritätsbedingungen. In diesem Abschnitt sollen einige weitere Ansätze und Korrektheitskriterien kurz vorgestellt werden.

a) Der bedingungsbasierte Ansatz des Demarkationsprotokoll

Einen weiteren Ansatz zur Durchsetzung globaler Integritätsbedingungen bildet das *Demarkationsprotokoll* aus [BG94]. In diesem Artikel wird davon ausgegangen, daß globale Integritätsbedingungen in der Regel recht einfacher Natur sind und vom GTM direkt durchgesetzt werden können, ohne dabei die Serialisierbarkeit der Abarbeitung beachten zu müssen. Weiterhin wird der Anspruch erhoben, daß globale Integritätsbedingungen dazu tendieren, Anforderungen nur "näherungsweise" auszudrücken, so daß dem GTM flexiblere Möglichkeiten der Umsetzung gegeben werden.

Die Arbeitsweise des Demarkationsprotokolls läßt sich am besten anhand eines Beispiels demonstrieren. Man stelle sich beispielsweise ein aus zwei Datenbankknoten bestehendes föderiertes System mit einem Datenobjekt a im Knoten s_1 und einem Datenobjekt b im Knoten s_2 vor. Es gelte die globale Integritätsbedingung $a + b \geq 100$. Möchte nun eine Transaktion den Wert eines der beiden Datenobjekte verringern, so muß in konventionellen Systemen jedesmal der Wert des jeweils anderen Datenobjektes gelesen werden, um zu überprüfen, ob eine Verringerung die globale Integritätsbedingung verletzt. Es ist somit eine globale Transaktion nötig, um lokal eine solche Verringerung durchzuführen. Zusammen mit der Anwendung eines 2PC-Protokolls zur Sicherung der Transaktionsterminierung sind somit zwei Kommunikationsrunden nötig, um diese Reduzierung unter

Berücksichtigung der globalen Bedingung durchzuführen. Das Demarkationsprotokoll soll diesen hohen Nachrichtenaufwand durch die Definition lokaler *"Limits"*, die anstelle der zu manipulierenden Daten zur Formulierung der globalen Integritätsbedingung herangezogen werden, entscheidend verringern.

Für den dargestellten Fall lassen sich lokal die Objekte a_l und b_l in den Knoten s_1 bzw. s_2 definieren, die als Limits der Werte a und b fungieren. Lokal gelten dann die Integritätsbedingungen $a \geq a_l$ und $b \geq b_l$ und global ist die Bedingung $a_l + b_l \geq 100$ zu erfüllen. Es ist klar ersichtlich, daß die Datenobjekte a und b nun lokal beliebig manipuliert werden können, wenn dabei die lokalen Integritätsbedingungen nicht verletzt werden. Eine Abfrage des jeweils anderen Datenobjektes ist nicht mehr nötig. Das Demarkationsprotokoll dient nun der Organisation von Änderungen der Limitwerte. Soll beispielsweise der Wert für a_l um einen bestimmten Betrag erhöht werden und ist dies unter Einhaltung der lokalen Bedingung möglich, so ist diese Erhöhung eine *sichere* Operation und kann lokal durchgeführt werden. Der Knoten s_2 wird über die Erhöhung informiert und kann seinerseits eine Verringerung von b_l um den entsprechenden Betrag vornehmen, ohne nochmals kommunizieren zu müssen. Soll dagegen der Limitwert a_l verringert werden, so ist diese Verringerung eine *unsichere* Operation, da sie eine Erhöhung von b_l im Knoten s_2 zur Erhaltung der globalen Bedingung nach sich ziehen kann. Diese Erhöhung ist jedoch nur möglich, wenn dadurch die lokale Bedingung des Knotens s_2 nicht verletzt wird. Folglich wird durch das Protokoll nur in dem Fall, in dem ein Knoten eine unsichere Operation durchführen möchte, der entsprechende korrespondierende Datenbankknoten um Erlaubnis gefragt.

In [BG94] wird das Protokoll für das geschilderte Beispiel einer arithmetischen Ungleichungsbedingung ausführlich erläutert und einige Strategien zur Auslösung des Protokolls und Bestimmung der Grenzwerte vorgestellt. Das Demarkationsprotokoll kann jedoch auch für arithmetische Gleichungsbedingungen, referenzielle Integritätsbedingungen, Schlüsselwertbedingungen oder Replikatsbedingungen verwendet werden, da sich diese geeignet durch arithmetische Ungleichungsbedingungen darstellen lassen, wie in [BG94] gezeigt wird. Das Demarkationsprotokoll realisiert keine global serialisierbaren Ausführungen, es geht lediglich von lokal serialisierbaren Schedules aus. Unter der Annahme der LSR-Eigenschaft sichert es jedoch die Umsetzung der lokalen bzw. gemischten Integritätsbedingungen. Auch wenn die globalen Schedules nicht der 2LSR-Eigenschaft genügen, werden die globalen Integritätsbedingungen durch den GTM und die entsprechenden Server mit Hilfe des Demarkationsprotokolls "manuell" realisiert.

Die bisher in diesem Kapitel vorgestellten Ansätze haben gezeigt, wie durch die Betrachtung von Mechanismen zur Einhaltung von Integritätsbedingungen alternative Korrektheitskriterien für föderierte Umgebungen definiert werden können. Alle vorgestellten Ansätze sicherten unter bestimmten Restriktionen die strenge Korrektheit der föderierten Datenbank. [BGS92] zeigen jedoch, daß strenge Korrektheit nicht unbedingt ausreichend ist, um in bestimmten Applikationen alle unerwünschten Abarbeitungen zu verhindern. Folgendes Beispiel verdeutlicht dieses Argument:

Beispiel 5.4 Gegeben sei ein FDBS in einem Geldinstitut, das die Einlagen und Transfers der Kunden verwaltet. Es bestehe aus zwei Knoten mit einem Konto a im Knoten s_1 und einem Konto b im Knoten s_2. Es gelte die Integritätsbedingung, daß kein Konto einen negativen Stand haben darf. Die Transaktion GT_1 transferiert nun 500 DM von Konto a nach b, wobei GT_1 nur ausgeführt wird, wenn der Wert des Kontos a 500 DM überschreitet. Die Transaktion GT_2 ist eine Kontrolltransaktion, die regelmäßig alle Konten durchläuft und deren Kontostand überprüft. Die Konten a und b enthalten 1000 DM und 500 DM als Startwerte:

$$S_1 : \ r_1(a = 1000)w_1(a = 500)r_2(a = 500)$$
$$S_2 : \ r_2(b = 500)r_1(b = 500)w_1(b = 1000)$$

Da keine negativen Kontostände auftreten und die Endzustände konsistent sind, ist die Ausführung streng korrekt, jedoch sieht die Kontrolltransaktion 500 DM zu wenig. Diese "Anomalie" kann für bestimmte Anwendungen der Grund dafür sein, daß die strenge Korrektheit der Ausführungen als zu realisierendes Kriterium aus Sicht der Anwendung nicht ausreichend ist. □

Folglich ist es in bestimmten Fällen sinnvoll, zusätzlich zur oder anstelle der strengen Korrektheit Mechanismen einzusetzen, die bestimmte Ausführungsfolgen vermeiden, ohne dabei Serialisierbarkeit zu erfordern. Ein solcher Mechanismus soll nun vorgestellt werden.

b) Epsilon-Serialisierbarkeit

Die *Epsilon-Serialisierbarkeit* ist ein alternatives Korrektheitskriterium, das in [PL91] vorgestellt wird, und das im Gegensatz zu den vorangegangenen Kriterien nicht nur die Sicherung von Integritätsbedingungen betrachtet. Die Epsilon-Serialisierbarkeit wird in [PL91] im Zuge der Untersuchung von Kontrollmechanismen für replizierte Datenobjekte in verteilten Systemen eingeführt. Das Grundproblem liegt dabei darin, daß atomare Änderungen aller Kopien eines Datenobjektes unter Anwendung des globalen Serialisierbarkeitskriteriums nur schwer zu realisieren sind und zu einer starken Verringerung der Parallelität der Abarbeitungen führen. Die Idee besteht nun darin, die Atomarität der Änderungsoperation eines replizierten Datenobjektes dadurch zu lockern, daß innerhalb des Zeitintervalls, in dem alle Replikate des Datenobjektes geändert werden, "read only"-Transaktionen die einzelnen Kopien lesen können. Die dabei auftretenden und akzeptierten Inkonsistenzen durch das Lesen noch nicht aktualisierter Kopien werden durch das Festlegen bestimmter Grenzen, d.h. durch die Limitierung der Anzahl lesender Transaktionen, eingeschränkt. Dieser ursprünglich zur Kontrolle von Datenobjektkopien gemachte Ansatz läßt sich auch für beliebige andere Problemstellungen im Bereich der Transaktionsverwaltung verallgemeinern, da durch ihn ein Mechanismus gegeben ist, der eine individuelle Steuerung des Korrektheits- und Parallelitätsgrades eines Schedules erlaubt.

[PL91] unterteilen die zu verarbeitenden Transaktionen in die Menge der "update"-Transaktionen U^{ET} und die Menge der "read only"-Transaktionen Q^{ET}. Die Menge U^{ET} erhält die Datenbankkonsistenz, d.h. durch die atomare Abarbeitung einer Transaktion aus dieser Menge wird keine Integritätsbedingung verletzt. Es wird angenommen, daß die "update"-Transaktionen in einem serialisierbaren Schedule ausgeführt werden, d.h. die Konsistenz der Datenbank wird durch die Schedules nicht gefährdet. Betrachtet man jedoch auch die "read only"-Transaktionen, so sind nichtserialisierbare Abarbeitungen erlaubt, solange die Anzahl der nichtserialisierbaren Konflikte begrenzt ist. Der folgende Schedule verdeutlicht die entscheidenden Aspekte:

$$S: \; w_2(x)r_3(x)r_3(y)c_3w_1(y)r_1(z)c_1w_2(z)c_2$$

Die Transaktionen T_1 und T_2 sind "update"-Transaktionen, während T_3 eine "read only"-Transaktion ist. Die Projektion auf die Menge der "update"-Transaktionen ist serialisierbar ($T_1 \rightarrow T_2$), betrachtet man jedoch den Schedule einschließlich der Transaktion T_3, so ist S nicht serialisierbar ($T_2 \rightarrow T_3 \rightarrow T_1 \rightarrow T_2$). Man sagt, T_2 *exportiert* einen Konflikt zur "read only"-Transaktion und T_3 *importiert* einen Konflikt. Für jede "read only"-Transaktion läßt sich nun ein *Importlimit* und für jede "update"-Transaktion ein *Exportlimit* definieren, die die maximale Anzahl von Konflikten angeben, in die eine Transaktion involviert sein darf. Formal läßt sich dieser Sachverhalt folgendermaßen definieren:

Definition 5.6 (Epsilon-Serialisierbarkeit) Ein Schedule S ist ϵ-seriell, wenn seine Projektion auf die "update"-Transaktionen seriell ist und die Anzahl der importierten Konflikte jeder "read only"-Transaktion nicht deren Importlimit übersteigt und die Anzahl der exportierten Konflikte jeder "update"-Transaktion nicht deren Exportlimit überschreitet. Ein Schedule ist ϵ-serialisierbar, wenn er äquivalent zu einen ϵ-seriellen Schedule ist. □

Aus der Definition wird klar, daß wenn die Limits aller Transaktionen auf Null gesetzt werden, ϵ-serialisierbare Schedules auch serialisierbar sind. In [PL91] sind einige Mechanismen zur Kontrolle von Datenobjektreplikaten aufgeführt, die die Epsilon-Serialisierbarkeit ausnutzen, die jedoch nicht Thema dieser Arbeit sein sollen.

Zusammenfassend läßt sich feststellen, daß es in föderierten Umgebungen mitunter sinnvoll sein kann, zur globalen Serialisierbarkeit alternative Korrektheitskriterien einzusetzen. Dies geschieht in erster Linie, um den lokalen Autonomieansprüchen der Systeme besser gerecht zu werden, das globale Transaktionsmanagement zu vereinfachen und die Parallelität der Abarbeitungen und den Transaktionsdurchsatz zu erhöhen. Hervorzuheben sind dabei die Kriterien QSR, 2LSR und LSR, die eine fortlaufende Abschwächung des globalen Serialisierbarkeitsbegriffes vornehmen. Für die Schedulemengen dieser Kriterien gilt die Inklusionsbeziehung $GSR \subset QSR \subset 2LSR \subset LSR$. Der Einsatz alternativer Kriterien hat jedoch zur Folge, daß mitunter nicht alle Arten von Integritätsbedingungen durch die Abarbeitungen realisiert werden können oder Zugriffsbeschränkungen

für bestimmte Transaktionen durchgesetzt werden müssen. Aus diesem Grund scheint die Umsetzung alternativer Ansätze nur in lose gekoppelten Umgebungen sinnvoll, da in diesen aufgrund der vielen unterschiedlichen Sichten auf die föderierte Datenbank eine einheitliche Umsetzung von Integritätsbedingungen nicht möglich erscheint.

An dieser Stelle soll die Untersuchung von globalen und lokalen Synchronisationsmechanismen abgeschlossen werden. Im Kapitel 4 wurde gezeigt, wie in fehler- und abbruchfreien Umgebungen globale Serialisierbarkeit erreicht werden konnte, einerseits auf Seiten des GTM durch Forcierung direkter Konflikte zwischen allen globalen Transaktionen, andererseits durch restriktivere Anforderungen an die lokalen Synchronisationsmechanismen. Im Kapitel 5 wurde untersucht, inwieweit man die Forderung nach globaler Serialisierbarkeit lockern kann, ohne dabei Abarbeitungen zu riskieren, die die Datenbankkonsistenz gefährden bzw. wie man gewisse inkonsistente Abarbeitungen zulassen kann, wenn bestimmte Grenzen nicht überschritten werden. Im folgenden Kapitel soll nun untersucht werden, wie in föderierten Umgebungen geeignet auf Fehler und Abbrüche reagiert werden kann.

Kapitel 6

Atomarität und Dauerhaftigkeit

In den vorangegangenen Kapiteln sind verschiedene Korrektheitskriterien und Umsetzungsmethoden für Synchronisationsmechanismen in FDBSen vorgestellt wurden. Dabei wurde stets angenommen, daß in einer fehler- und abbruchfreien Umgebung gearbeitet wird. In diesem Kapitel soll nun untersucht werden, wie die Atomarität von Transaktionen und die Konsistenz der Abarbeitung globaler Transaktionen in einer föderierten Datenbank erhalten werden kann, wenn Subtransaktionen abgebrochen werden oder während der Abarbeitung Fehler auftreten. Die Bandbreite der zu betrachtenen Situationen reicht dabei von einfachen unilateralen Transaktionsabbrüchen über System- und GTM-Fehler bis zu Verbindungs- und Kommunikationsfehlern. Besonders zu untersuchen sind dabei sogenannte *globale Transaktionsfehler*, bei denen eine globale Transaktion in einem Datenbankknoten durch ein unilaterales "abort" oder einen lokalen Fehler abgebrochen wird, während sie in einem anderen Knoten mit "commit" endet. Mechanismen, die solche Fehlerzustände und unilateralen Abbrüche behandeln, werden als *Recovery*- bzw. Rücksetzungsmechanismen bezeichnet. Im folgenden Kapitel soll davon ausgegangen werden, daß alle an einer Föderation partizipierenden CDBMSe über lokale Recovery-Mechanismen verfügen und so für die Atomarität und Dauerhaftigkeit der lokalen Transaktionen und globalen Subtransaktionen eines Knotens sorgen. Damit reduziert sich die Aufgabenstellung der allgemeinen Gewährleistung von Atomarität und Dauerhaftigkeit der Transaktionen auf die Sicherung einheitlicher Endzustände globaler Transaktionen, d.h. es muß garantiert werden, daß jede globale Transaktion in allen involvierten Datenbankknoten entweder mit "commit" abgeschlossen wird oder mit "abort" abbricht.

Die Sicherung der Atomarität von globalen Transaktionen in Multidatenbankumgebungen gilt in der Regel als sehr schwierig zu lösendes Problem. In [MKSA92] wird sogar bewiesen, daß bei der Einhaltung *vollständiger Autonomie* der lokalen Systeme die Gewährleistung der atomaren Ausführung einer globalen Transaktion *unmöglich* ist. Der Grund für diese Unmöglichkeit liegt in der Definition der lokalen Autonomie der CDBMSe. Die lokalen DBMSe gelten in [MKSA92] als autonom, wenn sie lokal die ACID-Eigenschaften sichern, d.h. es wird davon ausgegangen, daß die Scheduler serialisier- und rücksetzbare Abarbeitungen erzeugen. Folglich kann jeder lokale Scheduler, der diese Ei-

genschaften realisiert, eingesetzt werden. Weiterhin kann ein CDBMS nicht zwischen globalen und lokalen Transaktionen unterscheiden. Die lokalen DBMSe stellen dem GTM lediglich die üblichen Transaktionsoperationen an den lokalen Schnittstellen zur Verfügung, jedoch keine "prepare-to-commit"- oder "service-request"-Operationen (vergleiche Abschnitt 3.1). Folglich hat das globale Transaktionsmanagement keine direkte Kontrolle über die lokale Ausführung der Transaktionsoperationen, es ist ihm lediglich möglich, Kontrolloperationen, wie z.B. Ticketoperationen, im Zuge der Einreichung der globalen Transaktionsoperationen an die CDBMSe zu überreichen, um so eventuell eine "indirekte" Kontrolle zu erlangen. Abschließend wird verlangt, daß für jede globale Transaktion maximal eine Subtransaktion pro CDBS erzeugt wird. Unter diesen gegebenen Voraussetzungen ist es nicht möglich, die atomare Ausführung einer globalen Transaktion zu garantieren.

In [MKSA92] wird dies anhand eines einfachen Beispieles gezeigt. In diesem Beispiel wird eine einfache Transaktion in zwei Datenbankknoten einer Föderation ausgeführt. Die Subtransaktion im ersten Knoten wird erfolgreich beendet, im zweiten Knoten bricht der lokale Scheduler die Subtransaktion zum "commit"-Zeitpunkt zur Sicherung eines lokal serialisierbaren Schedules ab[1]. Aufgrund der lokalen Autonomie der partizipierenden Systeme ist es so möglich, daß eine globale Transaktion in einigen Knoten erfolgreich endet, in anderen Knoten dagegen abgebrochen wird. Eine solche partielle Ausführung einer globalen Transaktion kann jedoch globale Konsistenzverletzungen nach sich ziehen, so daß der GTM das Wiederherstellen eines konsistenten globalen Zustandes forcieren muß. Er kann dies durch ein Rücksetzen der erfolgreich beendeten Transaktionen oder ein Wiederholen der abgebrochenen Transaktionen erreichen. [MKSA92] zeigen jedoch für beide Fälle, daß es unter den gegebenen Umständen dem GTM nicht möglich ist, die Ausführung lokaler Transaktionen auf den global inkonsistenten Zuständen zu verhindern, so daß, in Abhängigkeit der Effekte der lokalen Transaktionen, das Wiederherstellen eines global konsistenten Zustandes unter Umständen unmöglich werden kann. Der Einsatz von S2PL-Schedulern in allen beteiligten CDBSen wird oftmals als akzeptable Verletzung der lokalen Autonomie angesehen. Diese Scheduler führen keine unilateralen Abbrüche zur Erhaltung der lokalen Serialisierbarkeit durch, jedoch können globale Subtransaktionen auch in diesem Fall durch lokale Systemfehler abgebrochen werden, so daß eine Situation ähnlich der zuvor geschilderten entsteht. In [MKSA92] kommt man somit zu dem Schluß, daß die Sicherung der Atomarität und Fehlertoleranz globaler Transaktionen in föderierten Umgebungen nur möglich ist, wenn man die Forderung der lokalen Autonomie "zurückschraubt", die Arten der erlaubten Transaktionen limitiert oder neue Transaktionsmodelle und Korrektheitskriterien verwendet. Nachfolgend sollen für jede dieser Strategien globale Recovery-Mechanismen vorgestellt werden.

Ein wesentlicher Faktor bei der Erstellung globaler Recovery-Mechanismen sind die zur Verfügung stehenden lokalen Schnittstellen. Steht beispielsweise in allen Systemen ein

[1] In diesem Beispiel wird ein "2PL-Certifier" [BHG87] verwendet, der eine Validierung zum "commit"-Zeitpunkt durchführt und in diesem Beispiel die Subtransaktion abbricht, da ein "commit" die lokale Serialisierbarkeit gefährden würde. Jeder andere lokale Scheduler, der die Ausführung einer Transaktion ebenfalls zum Terminierungszeitpunkt validiert, kann zu einem ähnlichen Ergebnis kommen.

sichtbarer "prepared-to-commit"-Zustand zur Verfügung, kann ein atomares "commit"-Protokoll (AC-Protokoll), wie das schon mehrfach erwähnte 2PC-Protokoll, zur Realisierung der Aufgabenstellung zur Anwendung kommen. Steht ein solcher Zustand nicht zur Verfügung, sind andere Techniken anzuwenden, um zu verhindern, daß eine globale Transaktion mit verschiedenen lokalen Endzuständen endet. Drei allgemeine Herangehensweisen werden dabei unterschieden:

- **Redo-Ansatz:** Nach einem Abbruch einer Subtransaktion in einem Knoten wird eine Redo-Transaktion abgearbeitet, die aus allen "write"-Operationen der abgebrochenen Subtransaktion besteht.
- **Retry-Ansatz:** Die abgebrochene Subtransaktion wird vollständig wiederholt abgearbeitet.
- **Compensate-Ansatz:** Wird eine globale Transaktion abgebrochen, so wird in jedem Knoten, in dem eine Subtransaktion der globalen Transaktion bereits die "commit"-Operation ausgeführt hat, eine kompensierende Subtransaktion ausgeführt, die die Effekte der beendeten Subtransaktion zurücksetzt.

Prinzipiell ist jeder der aufgeführten Ansätze mit jedem Synchronisationsmechanismus unter Einhaltung bestimmter zusätzlicher Bedingungen kombinierbar, jedoch haben die meisten Autoren bestimmte Präferenzen. So wird beispielsweise schon bei den allgemeinen Voraussetzungen für die Ticketmethoden aus [GRS91, GRS94] die Verwendung des 2PC-Protokolls nahegelegt und alle Korrektheitsbeweise beziehen sich auf die Anwendung dieses Recovery-Protokolls. Sowohl das 2PC-Protokoll als bekanntestes AC-Protokoll als auch die drei oben aufgeführten Ansätze werden nachfolgend vorgestellt, und es wird erläutert, unter welchen Bedingungen sie mit bestimmten Synchronisationsmechanismen kombiniert werden können.

6.1 Das 2PC-Protokoll

Das *Zwei-Phasen-Commit-Protokoll* (2PC-Protokoll) ist das bekannteste atomare "commit"-Protokoll und wird in einer ganzen Reihe von Publikationen näher vorgestellt [BHG87, VG93, Rah94]. Es stammt ursprünglich aus dem Bereich der verteilten Datenbanken, wird aber unter Annahme bestimmter Voraussetzungen auch in vielen föderierten Transaktionsverwaltungsansätzen zu Recovery-Zwecken verwendet, z.B. bei den Ticketmethoden von [GRS91, GRS94] in Abschnitt 4.1. Die grundlegende Voraussetzung für die Anwendung dieses Protokolls ist dabei die Unterstützung einer "prepare-to-commit"-Operation bzw. eines sichtbaren "prepared-to-commit"-Zustandes durch die Schnittstellen der lokalen DBMSe.

Ist diese Voraussetzung erfüllt, kann das Protokoll zur Anwendung kommen. Die Arbeitsweise des 2PC-Protokolls wird in der Abbildung 6.1 dargestellt. Der GTM fungiert als Koordinator der Abarbeitung und die CDBMSe bilden die Teilnehmer, die Endzustände der globalen Transaktion bzw. der einzelnen Subtransaktionen auf Koordinator-

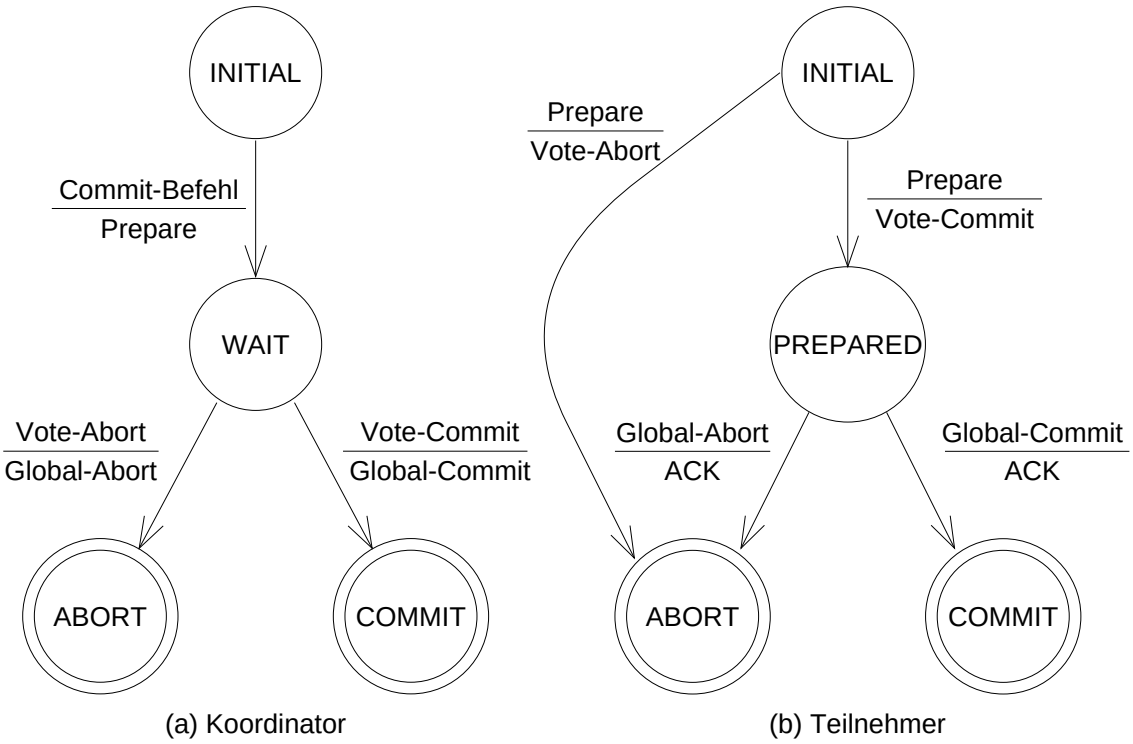

Abbildung 6.1: Zustandsübergänge beim 2PC-Protokoll

bzw. Teilnehmerseite sind durch einen Doppelkreis gekennzeichnet. Zur Terminierung der Abarbeitung einer globalen Transaktion schickt der GTM an alle lokalen Datenbankknoten, auf denen die globale Transaktion gearbeitet hat, eine "prepare-to-commit"-Operation. Nachdem ein lokales DBMS eine solche Operation empfangen hat, gibt es ein Votum ab, ob die entsprechende Transaktion mit "commit" beendet oder mit "abort" abgebrochen werden soll. Stimmt das lokale System für ein "commit" der globalen Transaktion, tritt die entsprechende Subtransaktion in den "prepared"-Zustand ein. Durch den Eintritt in diesen Zustand verliert das lokale DBMS das Recht, die Subtransaktion unilateral abzubrechen. Der GTM sammelt alle Stimmen der involvierten Datenbanksysteme und entscheidet daraufhin über die Terminierung der globalen Transaktion. Stimmten alle beteiligten Systeme für "commit", so entscheidet sich auch der GTM für "commit" und schickt die entsprechende Operation an die lokalen DBMSe. Ist dagegen in mindestens einem lokalen Datenbanksystem der "prepared"-Zustand nicht rechtzeitig erreicht worden, sei es durch ein "abort"-Votum des entsprechenden Systems oder durch den Ablauf des Timeout-Intervalls bei der Abfrage der einzelnen Stimmen, wird die globale Transaktion abgebrochen und ein "abort" an alle Knoten geschickt, die für "commit" gestimmt haben. Die CDBMSe sind dann gezwungen, die globale Entscheidung lokal umzusetzen.

Mit dem Eintritt einer Transaktion in den "prepared"-Zustand muß das lokale System in der Lage sein, sowohl die Transaktion ordnungsgemäß zu beenden als auch sie

abzubrechen. Da dabei auch Situationen zu berücksichtigen sind, in denen ein Systemfehler auftritt, während eine Transaktion im "prepared"-Zustand verweilt, müssen die durchgeführten Modifikationen schon mit dem Eintritt in den Zustand in den stabilen Datenbankspeicher geschrieben werden, um beim Wiederanlauf des Systems berücksichtigt werden zu können. Auf der anderen Seite muß ebenfalls sichergestellt werden, daß bei einer globalen Abbruchentscheidung dieses frühzeitige Festschreiben der Transaktionsergebnisse nicht die allgemeine Abarbeitungskonsistenz im lokalen System gefährdet und das CDBMS in der Lage ist, der globalen Entscheidung zu folgen.

Zur Sicherung dieser Eigenschaft ist es nötig, daß in allen Knoten s_k, auf denen eine Transaktion T_i gearbeitet hat, die Menge der Transaktionen $\mathcal{T}_k$, von denen T_i gelesen hat, bereits "committed" ist, wenn T_i in den "prepared"-Zustand eintritt. Ansonsten wäre es möglich, daß durch den Abbruch einer Transaktion aus $\mathcal{T}_k$ auch T_i abbrechen muß, so daß eine mögliche globale "commit"-Entscheidung lokal nicht mehr umgesetzt werden kann. Diese Anforderung ist erfüllt, wenn alle lokalen DBMSe mit Schedulern ausgestattet sind, die serialisierbare (SR) und rücksetzbare (RC) Schedules erzeugen. Die vorgestellten Synchronisationsmechanismen aus Abschnitt 4.2.3 sicherten streng rücksetzbare Abarbeitungen und damit sowohl die SR- als auch RC-Eigenschaft der lokalen Schedules. Sichern die lokalen Scheduler außerdem nichtkaskadierende Abarbeitungen [BHG87, VG93], kann der GTM die globale "prepare"-Operation senden, sobald die Subtransaktionen in allen Knoten ihre r/w-Operationen beendet haben, da obige Bedingung dann ebenfalls erfüllt ist [BGS92].

In [MR91] wird das 2PC-Protokoll dadurch charakterisiert, daß die globale Entscheidung *"in der Mitte"* der lokalen "commit"-Prozeduren stattfindet. Die lokalen "commit"-Prozeduren starten mit dem Eintreffen der "prepare"-Operation und dem Eintritt der Subtransaktionen in den "prepared"-Zustand. Alle dabei durchgeführten Aktionen entsprechen denen bei einem "commit" in einem nichtverteilten System. Danach wird die lokale "commit"-Prozedur unterbrochen, um eine globale Terminierungsentscheidung zu finden. Wurde durch den GTM eine Entscheidung getroffen, wird diese durch die lokalen "commit"-Prozeduren umgesetzt, indem z.B. im Falle einer globalen "commit"-Entscheidung und eines sperrenden Schedulers der finale "commit"-Zustand eingenommen wird und die Sperren der Subtransaktion freigesetzt werden. Aus Sicht der einzelnen DBMSe findet der globale Entscheidungsprozeß in der Mitte der lokalen "commit"-Prozeduren statt. Dieser Sachverhalt wird in der Abbildung 6.2 zum Ausdruck gebracht.

Die Anwendung des 2PC-Protokolls im Kontext der föderierten Datenbanksysteme bringt eine Reihe von Problemen mit sich, die vor allem in der Heterogenität der Systeme begründet liegen. Da es keinen einheitlichen Standard für dieses Protokoll gibt, sind so lokal unterschiedliche Implementierungen möglich, die die Fehlerbehandlung oder die Kontrolle des globalen "commits" betreffen. Weiterhin können bestimmte Knoten auf die Unterstützung erweiterter AC-Protokolle, wie z.B. das 3PC-Protokoll ("three phase commit protocol") [BHG87], vorbereitet sein, während andere überhaupt keine derartige Unterstützung liefern können. Letzteres ist z.B. der Fall, wenn ein lokaler Knoten nur "Service Requests" an der lokalen Schnittstelle bereitstellt (vergleiche Abschnitt 3.1)

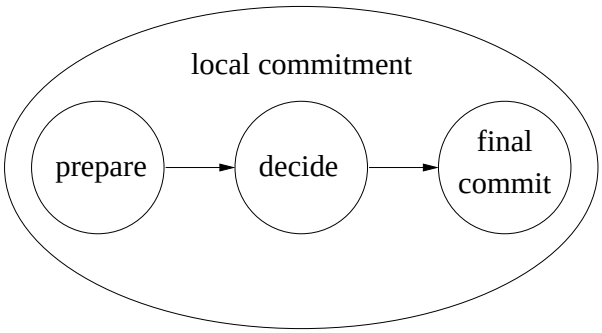

Abbildung 6.2: Entscheidungsfindung *in der Mitte* der lokalen "commit"-Prozedur

oder seine Ausführungs- bzw. Kommunikationsautonomie behalten möchte. Viele Publikationen im Bereich der föderierten Datenbanksysteme unterstreichen die laufende Debatte über die Unterstützung der "prepare-to-commit"-Operationen bzw. "prepared"-Zustände durch die lokalen Systeme und damit die Anwendbarkeit des 2PC-Protokolls. Einerseits wird durch die Anwendung des Protokolls der Kreis der in eine Föderation integrierbaren Systeme begrenzt und die Autonomie der Komponenten eingeschränkt, andererseits liefert das 2PC-Protokoll bewährte Mechanismen zur Realisierung verteilter Recovery-Mechanismen.

Viele Datenbanksysteme, die eine Client-Server-Architektur besitzen, wie z.B. Sybase, unterstützen sichtbare "prepared"-Zustände und können so jederzeit an einem FDBS mit 2PC-Recovery teilnehmen. Andererseits können Systeme ohne sichtbare "prepared"-Zustände diese geeignet simulieren, wie [GRS91, GRS94] argumentieren. Handshake-Signale nach jeder Operation oder das asynchrone Einreichen von Operationen und Abfragen des Abbruchstatus nach dem RDA-Standard sind solche Simulationsmethoden.

Für Systeme, in denen die Anwendung eines globalen atomaren "commit"-Protokolls dennoch nicht möglich oder erwünscht ist, werden in den nachfolgenden Abschnitten alternative Recovery-Mechanismen vorgestellt.

6.2 Der Redo-Ansatz

Der *Redo-Ansatz* ist der erste Ansatz, der davon ausgeht, daß die lokalen Datenbankmanagementsysteme keine "prepare-to-commit"-Operationen unterstützen. Um in solchen Systemen die Atomarität globaler Transaktionen zu sichern, kann das 2PC-Protokoll angewandt werden, wenn anstatt der CDBMSe die Server als Teilnehmer am Protokoll fungieren. In diesem Fall sind die lokalen DBMSe in der Lage, Subtransaktionen jederzeit abzubrechen, also auch nachdem der Server beispielsweise für "commit" im 2PC-Protokoll gestimmt hat. Wurde eine solche Subtransaktion unilateral abgebrochen und hat sich der GTM für ein globales "commit" entschieden, obliegt es dem Server des entsprechenden Datenbankknotens, für einen Datenbankzustand zu sorgen, der einer atomaren Terminierung der globalen Transaktion entspricht. Der Server startet dazu im

lokalen Datenbanksystem eine *Redo-Transaktion*, die aus allen *write*-Operationen der abgebrochenen Subtransaktion besteht. Zur Erzeugung einer solchen Redo-Transaktion verwaltet der Server eine sogenannte *"server log"*-Datei, die alle "update"-Operationen globaler Transaktionen enthält. Treten bei der Ausführung der Redo-Transaktion Fehler auf, wird diese Transaktion so lange durch den Server wiederholt, bis sie im "committed"-Zustand endet.

Wird ein solcher Ansatz zur Sicherung der Atomarität der globalen Transaktionen gewählt, ist die Realisierung bestimmter lokaler Scheduleeigenschaften nötig. So muß gewährleistet sein, daß ein Server sein "commit"-Votum für eine Transaktion T_i nur dann zum GTM schickt, wenn in dem entsprechenden Knoten s_k die Menge der Transaktionen $\mathcal{T}_k$, von denen T_i gelesen hat, bereits "committed" ist. Ansonsten ist es wie beim vorangegangenen Ansatz möglich, daß durch den Abbruch einer Transaktion aus $\mathcal{T}_k$ auch T_i abbrechen muß, obwohl eine globale "commit"-Entscheidung getroffen wurde und so die Entscheidung des GTM nicht entsprechend umgesetzt werden kann. Wenn T_i nur von globalen Transaktionen liest, kann der Server diese Eigenschaft sichern, indem er sein Votum so lange verzögert, bis alle Transaktionen aus $\mathcal{T}_k$ mit "commit" beendet wurden. Da jedoch T_i prinzipiell auch von lokalen Transaktionen lesen kann und der Server keine Kontrolle über die Ausführung lokaler Transaktionen hat, kann die Eigenschaft nur gesichert werden, wenn das lokale DBMS einen Scheduler benutzt, der garantiert, daß in den Schedules keine kaskadierenden Abbrüche vorkommen. Folglich fordert die Anwendung des Redo-Ansatzes, daß in jedem lokalen DBMS ein ACA-Scheduler zum Einsatz kommt.

Beispiel 3.3 verdeutlichte bereits ein weiteres Problem des Redo-Ansatzes: Aus Sicht der jeweiligen lokalen DBMSe stellen die Redo-Transaktionen eigenständige Transaktionen dar und sie werden entsprechend der lokalen Transaktionsverwaltungsmechanismen auch als solche behandelt. Aus Sicht des GTM sind sie jedoch Teil der globalen Transaktion, so daß lokal erzeugte Schedules aus der Sicht des föderierten Transaktionmanagements nicht serialisierbar sein können. Dies widerspricht jedoch sämtlichen Korrektheitskriterien der Kapitel 4 und 5, die alle von ausgingen, daß lokale Schedules aus globaler Sicht serialisierbar sein müssen. In [MRB+92a] wurde für Schedules, die aus Sicht des GTM serialisierbar sind, das Kriterium der *M-Serialisierbarkeit* eingeführt:

Definition 6.1 (M-Serialisierbarkeit) Es sei S_k ein lokaler Schedule, der aus den lokalen Transaktionen, den globalen Subtransaktionen und den Redo-Transaktionen einen Knotens s_k besteht. Es sei $m(S_k)$ eine Projektion von S_k auf die Menge der "commit"-abgeschlossenen Transaktionen und der "read"-Operationen von globalen Transaktionen, die vom lokalen Transaktionsmanagement abgebrochen, durch den GTM aber mit "commit" beendet wurden. Es sei T_i eine solche lokal abgebrochene globale Subtransaktion. In $m(S_k)$ werden die "read"-Operationen von T_i und die "write"-Operationen der zu T_i gehörenden Redo-Transaktion als eine Transaktion betrachtet. Ein Schedule S_k heißt genau dann *m-serialisierbar*, wenn $m(S_k)$ serialisierbar ist. □

Offensichtlich ist der Schedule aus Beispiel 3.3 nicht m-serialisierbar, da die Projektion des Schedules S_1 wie folgt aussieht:

$$m(S_1): \ r_1(a)r_2(a)w_2(a)c_2w_1(a)c_1$$

Um zu gewährleisten, daß die mit der Redo-Technik realisierten Ausführungen eine Datenbank von einen konsistenten Zustand in einen konsistenten Zustand überführt, ist es somit notwendig, die Redo-Technik mit den Techniken zur Realisierung von m-serialisierbaren Ausführungen zu kombinieren. Eine solche Technik wäre das Aufstellen bestimmter Restriktionen für die Daten, auf die lokale und globale Transaktionen zugreifen dürfen. Geht man dabei wie bereits im Abschnitt 5.2 von einer Unterteilung der Datenobjekte in eine Menge lokaler und eine Menge globaler Objekte aus, dann läßt sich durch folgende Bedingung das Problem aus Beispiel 3.3 verhindern:

> *Globale Transaktionen, die im Knoten s_k lokale Datenobjekte gelesen haben, sind nicht berechtigt, lokale Datenobjekte in s_k, auf die durch lokale Transaktionen zugegriffen werden kann, zu schreiben.*

Gilt die aufgestellte Restriktion, so läßt sich die nicht m-serialisierbare Ausführung aus Beispiel 3.3 vermeiden. In diesem Fall ist a ein lokales Datenobjekt, da es von der lokalen Transaktion LT_2 geschrieben wird. Da auch die globale Transaktion GT_1 das Datenobjekt a liest und schreibt, ist obige Bedingung verletzt und S_1 ist nicht m-serialisierbar.

Leider ist jedoch die obige Restriktion nicht ausreichend, wie [MRB+92a] zeigen. Es muß vielmehr gefordert werden, daß die lokalen Schedules außer der ACA-Eigenschaft auch streng rücksetzbar sein müssen. Außerdem fordern [MRB+92a], daß der GTM sicherstellt, daß die Projektion des globalen Schedules auf die Menge der globalen Transaktionsoperationen rigoros sein muß.

[MRB+92a] zeigen aber auch, daß die Forderung nach streng rücksetzbaren und nicht kaskadierend abbrechenden lokalen Schedules, sowie die Beschränkung auf rigorose Projektionen globaler Transaktionen gelockert werden kann, wenn weitere Restriktionen an die durch globale Transaktionen verarbeiteten Datenobjekte gemacht werden:

> *Globale Transaktionen, die im Knoten s_k lokale Datenobjekte gelesen haben, sind nicht berechtigt, Datenobjekte beliebiger Art in s_k zu modifizieren.*

Nachdem zunächst gezeigt wurde, wie M-Serialisierbarkeit in der Gegenwart von lokalen Transaktionsabbrüchen und Fehlern realisiert werden kann, wenn zur Sicherung der atomaren Ausführung globaler Transaktionen Redo-Techniken eingesetzt werden, soll nun untersucht werden, wie dieser Ansatz mit den Korrektheitskriterien aus den Kapiteln 4 und 5 kombiniert werden kann. Man könnte erwarten, daß das Kombinieren der Mechanismen zur Durchsetzung eines Korrektheitskriteriums aus den vorangegangenen Kapiteln mit den Mechanismen zur Sicherung der M-Serialisierbarkeit ausreichend ist, um das gewünschte Kriterium auch in fehler- und abbruchbehafteten Umgebungen durchzusetzen.

Dies gilt jedoch nur für die alternativen Kriterien LSR und 2LSR. Da die M-Serialisierbarkeit lokaler Schedules deren Serialisierbarkeit impliziert, ist die Sicherung der M-Serialisierbarkeit in allen lokalen Systemen ausreichend zur Realisierung der LSR-Eigenschaft globaler Schedules. Soll die 2LSR-Eigenschaft der globalen Schedules sichergestellt werden, muß laut Definition der zweistufigen Serialisierbarkeit neben der lokalen M-Serialisierbarkeit die Serialisierbarkeit der Projektion des globalen Schedules auf die Menge der zu globalen Transaktionen gehörenden Operationen, die nachfolgend durch GS bezeichnet wird, gesichert werden. Es wurde bereits festgestellt, daß in Systemen, die die schwächere erste der oben aufgeführten Restriktionen erfüllen, zur Sicherung der M-Serialisierbarkeit, die Rigorosität von GS durch den GTM zu sichern ist. Nun impliziert die Rigorosität von GS auch die Serialisierbarkeit dieser Projektion, so daß die 2LSR-Eigenschaft der globalen Schedules in diesem Fall gesichert ist. Wird jedoch in allen Komponenten eines föderierten Systems die stärkere zweite Restriktion erfüllt, bleibt die Forderung nach der Serialisierbarkeit von GS zur Realisierung der 2LSR-Eigenschaft ein durch den GTM zu lösendes Problem.

Soll die globale Serialisierbarkeit als Korrektheitskriterium eines föderierten Systems realisiert werden, reicht es dagegen nicht aus, die Synchronisationsmechanismen nur um Techniken zur Durchsetzung von M-Serialisierbarkeit zu erweitern, um Fehler und Abbrüche in einem solchen System korrekt zu behandeln. Zur Illustration betrachte man ein föderiertes System, in dem alle lokalen Scheduler rigorose Abarbeitungen garantieren. In diesem Fall sichern "commitverzögerte" globale Transaktionen die globale Serialisierbarkeit, wie bereits im Abschnitt 4.2.4 festgestellt wurde. Zur Sicherung der M-Serialisierbarkeit in dieser Umgebung ist es ausreichend, die Rigorosität von GS zu sichern. Dies ist jedoch nicht ausreichend, um die globale Serialisierbarkeit in der Gegenwart von Fehlern und Transaktionsabbrüchen zu gewährleisten, wie folgendes Beispiel demonstriert:

Beispiel 6.1 Ein FDBS bestehe aus zwei Knoten: s_1 mit den globalen Datenobjekten a und b, und s_2 mit den globalen Datenobjekten c und d. Es seien GT_1 und GT_2 globale Transaktionen und LT_3 und LT_4 lokale Transaktionen auf den Knoten s_1 bzw. s_2:

$$\begin{aligned} GT_1 &: \ w_1(a)w_1(c) \\ GT_2 &: \ w_2(b)w_2(d) \\ LT_3 &: \ r_3(a)r_3(b) \\ LT_4 &: \ r_4(c)r_4(d) \end{aligned}$$

Betrachtet werden soll folgende Situation: Der GTM hat sich für ein "commit" von GT_1 und GT_2 entschieden, jedoch wurde GT_1 in s_1 durch das lokale Transaktionsmanagement abgebrochen. Folglich muß in s_1 die folgende Redo-Transaktion RT_5 abgearbeitet werden, um die von GT_1 durchgeführten Änderungen durchzusetzen:

$$RT_5 : \ w_5(a)$$

Damit ergeben sich die folgenden lokalen Schedules in s_1 bzw. s_2:

$$S_1 : \ w_1(a)a_1w_2(b)c_2r_3(a)r_3(b)c_3w_5(a)c_5$$
$$S_2 : \ w_1(c)c_1r_4(c)r_4(d)c_4w_2(d)c_2$$

S_1 und S_2 sind m-serialisierbar (in S_2 sind alle Transaktionen mit "commit" beendet wurden, so daß aus der Serialisierbarkeit trivialerweise die M-Serialisierbarkeit folgt; auch S_1 ist m-serialisierbar, da bei der Betrachtung von $m(S_1)$ die Operation $w_1(a)$ herausfällt). Trotzdem sind die obigen Ausführungen nicht global serialisierbar (in s_1 gilt die Abhängigkeit $GT_2 \rightarrow LT_3 \rightarrow GT_{1(5)}$, wogegen in s_2 die Abhängigkeit $GT_1 \rightarrow LT_4 \rightarrow GT_2$ gilt). □

Das Problem bei diesem Beispiel liegt darin, daß globale Transaktionen, die in der ursprünglichen Verarbeitung weder in einem direkten noch einem indirekten Konflikt zueinander standen, durch einen Fehler mit nachfolgender Redo-Transaktion in einen indirekten Konflikt geraten können. Eine mögliche Lösung für dieses Problem ist die völlige Entkopplung der globalen und lokalen Daten, d.h. globale Transaktionen dürfen nicht auf lokale Datenobjekte zugreifen und lokalen Transaktionen ist der Zugriff auf globale Daten untersagt. Ist dies gewährleistet, so ist es mit der Sicherung m-serialisierbarer lokaler Schedules dem GTM möglich, globale Serialisierbarkeit zu gewährleisten.

Nachfolgend sollen mit dem "Commit"-Graphen-Ansatz und der "2PC-Agent"-Methode zwei alternative Möglichkeiten zur Sicherung der globalen Serialisierbarkeit in fehler- und abbruchanfälligen Systemen vorgestellt werden, die eng mit den bisher gemachten Ausführungen in Verbindung stehen.

a) Der "Commit"-Graphen-Ansatz

In [BST90, BST92] wird ein graphenbasierter Ansatz zur Lösung des Recovery-Problems mit Hilfe des Redo-Ansatzes und zur allgemeinen Vermeidung globaler Verklemmungssituationen vorgestellt. Zunächst sollen hier die Recovery-Mechanismen aus [BST90, BST92] erläutert werden, bevor in Kapitel 7 auf die Mechanismen zur Behandlung globaler Verklemmungen eingegangen wird.

[BST90, BST92] gehen in ihren Ausführungen von einem Transaktionsmodell in FDBSen aus, das im wesentlichen dem in Abschnitt 3.2 dargestellten Ansatz entspricht. Jedoch werden einige Einschränkungen im Hinblick auf die Eigenschaften der CDBMSe gemacht und Modifikationen der Struktur des globalen Transaktionsmanagements vorgenommen. Das Modell von [BST90, BST92] basiert auf folgenden Annahmen:

- Es können keine Veränderungen an der lokalen DBMS-Software vorgenommen werden. Diese Annahme beruht auf der allgemeinen Autonomieforderung und ist wesentlicher Grund dafür, daß der GTM sich zwar des Faktes bewußt ist, daß in den einzelnen Knoten lokale Transaktionen ablaufen können, ihm jedoch keine Informationen über deren Abarbeitungszustände oder von ihnen referenzierte Datenobjekte zur Verfügung stehen.

- Die lokalen DBMSe sind nicht in der Lage, direkt miteinander zu kommunizieren. Vielmehr sind sich die CDBMSe nicht der Tatsache bewußt, in einer Föderation gemeinsam globale Transaktionen zu bearbeiten, und agieren unter der Illusion, daß jede bearbeitete Transaktion nur im Knoten des CDBMSs ausgeführt wird.
- Jedes lokale DBMS ist verantwortlich für die Atomarität der lokal verarbeiteten Transaktionen. Dies kann z.B. durch das Verwalten eines "write-ahead-log"-Schemas durch die CDBMSe realisiert werden, jedoch sind derartige Informationen nicht dem GTM zugänglich.
- Jedes lokale DBMS benutzt das strikte Zwei-Phasen-Sperr-Protokoll als lokalen Synchronisationsmechanismus und garantiert, daß keine lokalen Verklemmungen auftreten.
- Die CDBMSe sind nicht in der Lage zwischen lokalen und globalen Transaktionen zu unterscheiden, d.h. globale Anwendungen können keine Bevorzugung gegenüber lokalen Anwendungen verlangen.
- Die lokalen DBMSe können aufgrund ihrer Ausführungsautonomie nicht an einem atomaren "commit"-Protokoll, wie dem 2PC-Protokoll, teilnehmen.

Die Struktur des globalen Transaktionsmanagements unterscheidet sich von dem im Abschnitt 3.2 dargestellten Modell dadurch, daß der GTM in jedem Knoten für jede globale Transaktion einen eigenen Server anlegt, der als Agent der globalen Transaktion in diesem Datenbankknoten agiert und von den lokalen DBMSen als eine lokale Transaktion angesehen wird. Ein lokaler Server wird durch den GTM erst dann wieder freigegeben, wenn die globale Transaktion durch ein "commit" beendet oder mit "abort" abgebrochen wurde. Alle Operationen einer globalen Transaktion auf einem bestimmten Datenbankknoten werden über den entsprechenden Server zur Verarbeitung eingereicht. Dabei wird eine globale Transaktionsoperation (mit Ausnahme der ersten) erst dann durch den GTM zur Verarbeitung eingereicht, wenn er die Nachricht erhalten hat, daß die vorangegangene Operation erfolgreich abgearbeitet wurde. Zur Erstellung der Ausführungsreihenfolge der globalen Transaktionsoperationen verwaltet die Schedulerkomponente des GTM globale Sperren und benutzt ein S2PL-Protokoll (nach [BHG87]) zur Zuweisung der globalen Sperren. Fordern somit zwei globale Transaktionen gleichzeitig Sperren für dasselbe Datenobjekt an, die einen Konflikt erzeugen, d.h. zumindest eine Sperre ist eine Schreibsperre, so wird eine der beiden globalen Transaktionen in der Ausführung verzögert, um einen Konflikt in dem entsprechenden CDBS zu vermeiden. Es ist zu beachten, daß jede globale Transaktion, die eine lokale Sperre für ein Datenobjekt hält, auch die globale Sperre halten muß. Das Halten einer globalen Sperre impliziert jedoch nicht, daß die Transaktion auch die lokale Sperre des Datenobjekt hält, da sie mitunter in dem lokalen Datenbanksystem auf die Freigabe der entsprechenden Sperre durch lokale Transaktionen warten muß. Das Verwalten von globalen und lokalen Sperren in der dargestellten Art impliziert, daß die globalen und lokalen Sperren die gleiche Granularität haben.

Wie bereits in der allgemeinen Darstellung des Redo-Ansatzes ausgeführt, agieren die Server als Teilnehmer in einem 2PC-Protokoll und verwalten eine "Server-Log"-Datei, um jederzeit in der Lage zu sein, die Atomarität globaler Transaktionen durchzusetzen. Die Verarbeitung einer globalen Transaktion verläuft in der herkömmlichen Art und Weise: Der GTM reicht alle globalen Transaktionsoperationen an die Server zur Ausführung ein, sofern dadurch nicht die Bedingungen des globalen S2PL-Protokolls verletzt werden. Die Server reichen alle Operationen zur Ausführung an die lokalen DBMSe weiter und schreiben alle vorgenommenen Änderungen in ihre "Log"-Datei. Soll die globale Transaktion beendet werden, schickt der GTM eine "prepare-to-commit"-Nachricht an die Server, die mit einer "ready"-Nachricht[2] antworten, falls sie in der Lage sind, die globale Subtransaktion mit "commit" zu beenden. Antworten alle Server einer globalen Transaktion mit "ready", sendet der GTM eine "commit"-Nachricht an die Server, ansonsten wird ein "abort" an die Server geschickt. Die Server terminieren die Subtransaktionen in der vom GTM geforderten Weise.

Wie bereits erläutert, sind dabei besonders die Situationen problematisch, in denen aufgrund lokaler Fehlersituationen oder unilateraler Abbrüche eine Subtransaktion zwischen der "ready"- und der globalen "commit"-Nachricht abgebrochen wurde. Das Hauptproblem dieser Situation liegt darin, daß zwischen dem lokalen Abbruch und dem damit verbundenen Rücksetzen der lokalen Effekte einer globalen Transaktion in einem Knoten und der Ausführung der entsprechenden Redo-Transaktion durch den Server lokale und globale Transaktionen in diesem Knoten ausgeführt werden können, die die inkonsistenten Effekte der nichtatomar abgearbeiteten globalen Transaktion lesen können. Durch den Einsatz des globalen S2PL-Schedulers beschränkt sich das genannte Problem jedoch nur auf die lokalen Transaktionen. Die Ausführung von globalen Transaktionen, die ein, durch eine abgebrochene Subtransaktion modifiziertes Datenobjekt lesen sollen, wird nämlich durch den globalen Scheduler bis zur endgültigen Terminierung der ursprünglichen Transaktion, d.h. bis zur Vollendung der entsprechenden Redo-Transaktion, verzögert.

Zur Analyse des Problems werden in [BST90, BST92] die Mengen der gelesenen (*"read-set"*) und geschriebenen (*"write-set"*) Datenobjekte der einzelnen Transaktionen untersucht. Es sei dabei $\mathbf{read}(T)$ die Menge der von einer Transaktion T gelesenen Datenobjekte, dementsprechend sei $\mathbf{write}(T)$ die Menge der von T geschriebenen Datenobjekte. Weiterhin sei GT_i eine beliebige, im Knoten s_k ausgeführte globale Transaktion, $\mathcal{L}_k = \{LT_1, \ldots, LT_n\}$ sei die Menge der in s_k ausgeführten lokalen Transaktionen. Zur Vermeidung des in Beispiel 3.3 dargestellten Recovery-Problems ist es nach [BST92] aus-

[2]Zur Unterscheidung vom 2PC-Protokoll aus Abschnitt 6.1 wird hier von einer "ready"-Nachricht anstelle einer "vote-commit"-Nachricht bzw. einem "ready"-Zustand im Gegensatz zum "prepared"-Zustand gesprochen. Die Nachrichten bedeuten prinzipiell dasselbe, die Zustände unterscheiden sich jedoch dadurch, daß der "prepared"-Zustand im Gegensatz zum "ready"-Zustand vom lokalen DBMS zur Verfügung gestellt wird. Eine "prepared"-Transaktion kann nicht mehr durch das CDBMS unilateral abgebrochen werden. Der "ready"-Zustand ist dagegen ein "Pseudozustand": Aus Sicht des CDBMSs befindet sich die Transaktion in der Verarbeitung und wartet auf das Eintreffen der nächsten Transaktionsoperation, welche in diesem Fall die globale Terminierungsentscheidung ist. Das CDBMS ist dabei jederzeit in der Lage, diese Transaktion unilateral abzubrechen.

reichend, für jede globale Transaktion GT_i in jedem Knoten s_k eine der beiden folgenden Bedingungen zu erfüllen:

1. Bedingung
   - $\mathbf{write}(GT_i) \cap \mathbf{read}(LT_j) = \emptyset$ für alle $LT_j \in \mathcal{L}_k$
   - $\mathbf{write}(GT_i) \cap \mathbf{write}(LT_j) = \emptyset$ für alle $LT_j \in \mathcal{L}_k$
2. Bedingung
   - $\mathbf{read}(GT_i) \cap \mathbf{write}(LT_j) = \emptyset$ für alle $LT_j \in \mathcal{L}_k$

Im Beispiel 3.3 sind offensichtlich beide Bedingungen verletzt, da im Knoten s_1 die Mengen der durch GT_1 und LT_2 gelesenen und geschriebenen Objekte identisch sind.

Offensichtlich sind die aufgestellten Bedingungen unter den gegebenen Voraussetzungen ausreichend, um das globale Atomaritäts- und Fehlertoleranzproblem zu lösen. Benutzen alle lokalen Scheduler und die globale Schedulerkomponente S2PL-Methoden zur Erstellung der Ablaufpläne, so sind die erstellten lokalen Schedules und die Projektion des globalen Schedules auf die Menge der globalen Transaktionsoperationen rigoros. Demnach kann eine aus globaler Sicht nichtserialisierbare lokale Abarbeitung $GT_i \rightarrow LT_j \rightarrow GT_i$ im Knoten s_k nur dann entstehen, wenn die abgebrochene Subtransaktion von GT_i in s_k zunächst einige Datenobjekte gelesen hat, LT_j die durch GT_i gelesenen Werte nach dem Abbruch modifiziert und GT_i diese Datenobjekte anschließend während der Redo-Phase wieder schreibt. Bedingung 2 bricht offensichtlich die erste Abhängigkeit ($GT_i \rightarrow LT_j$) auf, wogegen Bedingung 1 die zweite Abhängigkeit ($LT_j \rightarrow GT_i$) aufhebt.

Zur Durchsetzung der oben genannten Bedingungen unterteilen [BST90, BST92] die Datenobjekte des FDBSs in disjunkte Mengen global (*"globally updateable"*) und lokal (*"locally updateable"*) modifizierbarer Objekte. Zur Durchsetzung von Bedingung 1 wird nun gefordert, daß lokale Transaktionen keine global modifizierbare Datenobjekte lesen oder schreiben dürfen und globale Transaktionen nur global modifizierbare Daten schreiben. Zur Durchsetzung von Bedingung 2 wird gefordert, daß globale "update"-Transaktionen keine lokal modifizierbaren Daten lesen dürfen. Es sei nochmals darauf hingewiesen, daß nur eine der beiden Bedingungen gelten muß, um konsistenzerhaltende Abarbeitungen zu sichern.

In [BST90, BST92] werden weiterhin die Bedingungen für eine korrekte Ausführung der globalen "commit"-Operation untersucht. Beispiel 6.1 hat bereits gezeigt, daß auch wenn alle bis hierher aufgeführten Bedingungen erfüllt sind, Situationen entstehen können, die die geforderte globale Serialisierbarkeit verletzen können. Da die globalen Transaktionen im Beispiel 6.1 nicht miteinander in Konflikt stehen, gibt es für die GTM keinen Grund, die Ausführung der Transaktionsoperationen zu verzögern. Der Abbruch einer globalen Subtransaktionen zusammen mit den indirekten Konflikten durch lokale Transaktionen ergab jedoch unterschiedliche lokale Serialisierungsreihenfolgen. Aus Sicht

des GTM kann diese Situation nur verhindert werden, wenn man die Ausführung der "commit"-Operation der globalen Transaktion GT_2 bis zur Vollendung des "commits" der Transaktion GT_1 verzögert, da durch die Rigorosität der lokalen Schedules dann die Abhängigkeit $GT_2 \rightarrow LT_3 \rightarrow GT_{1(5)}$ im Knoten s_1 vermieden wird.

Zur Realisierung einer derartigen "commit"-Verzögerung schlagen [BST90, BST92] den *"commit"-Graphen* (CG) vor. Der "commit"-Graph ist ein bipartiter ungerichteter Graph mit der Menge der lokalen Datenbanksysteme und der globalen Transaktionen als Knotenmenge. Der Graph enthält genau dann eine Kante (T_i, s_k) zwischen dem Knoten einer Transaktion und dem Knoten eines lokalen Systems, wenn T_i in s_k ausgeführt wird und die "commit"-Operation für T_i zur Abarbeitung eingereicht wurde. Nach der vollständigen Ausführung der "commit"-Operation von T_i wird der Knoten der Transaktion mit den angrenzenden Kanten aus dem Graphen entfernt. Für eine Menge globaler und lokaler Transaktionen ist unter den gegebenen Voraussetzungen deren Ausführung global serialisierbar, wenn der "commit"-Graph keine Schleifen enthält. Im Beispiel 6.1 wird diese Bedingung offensichtlich verletzt.

In [BST90, BST92] wird weiterhin die Nutzung eines *"wait-for-commit"-Graphen* ($WFCG$) vorgeschlagen. Der "wait-for-commit"-Graph ist ein gerichteter Graph mit der Menge der globalen Transaktionen als Knotenmenge. Der Graph enthält genau dann eine Kante $GT_i \rightarrow GT_j$ zwischen den Knoten zweier globaler Transaktionen, wenn GT_i die Ausführung der Datenbankoperationen abgeschlossen hat, die Ausführung der "commit"-Operation von GT_i aber verzögert wird und wenn GT_j eine globale Transaktion ist, deren "commit"-Operation vor dem "commit" von GT_i ausgeführt oder abgebrochen werden soll.

Mit Hilfe dieser beiden Graphen ist es dem GTM möglich zu entscheiden, ob die Ausführung einer globalen "commit"-Operation sicher im Hinblick auf die Konsistenzerhaltung der Abarbeitung ist. Der folgende Algorithmus dient dann dem GTM zur Konstruktion des CG bzw. $WFCG$ und zur Entscheidung, ob die Ausführung der "commit"-Operation einer Transaktion GT_i sicher ist:

1. Füge für jeden Knoten s_k, in dem GT_i ausgeführt wird, die temporäre Kante (T_i, s_k) in den "commit"-Graphen CG.

2. Enthält der so erweiterte CG keine Zyklen, wird die globale "commit"-Operation für GT_i zur Verarbeitung eingereicht und die temporären Kanten werden zu permanenten Kanten.

3. Enthält der erweiterte CG Zyklen, dann:

   (a) Es sei $\{GT_{i_1}, \ldots, GT_{i_n}\}$ die Menge all jener globalen Transaktionen, die im CG durch das Einfügen der temporären Kanten in einem Zyklus enthalten sind. Für jede Transaktion GT_{i_j} dieser Menge wird die Kante $GT_i \rightarrow GT_{i_j}$ in den $WFCG$ eingefügt.

   (b) Alle temporären Kanten werden aus dem CG gelöscht.

Zum $WFCG$ muß bemerkt werden, daß nicht notwendigerweise alle Transaktionen GT_{i_j}, für die eine Kante $GT_i \rightarrow GT_{i_j}$ existiert, ihre "commit"-Operationen ausgeführt haben müssen, bevor die "commit"-Operation von GT_i ausgeführt werden kann. In der Regel reicht es aus, wenn eine Teilmenge der angegeben Transaktionen (mitunter reicht sogar eine einzige) vor dem "commit" von GT_i beendet wird, um einen Zyklus im CG zu vermeiden.

Damit sollen zunächst die Ausführungen zu dem in [BST90, BST92] vorgestellten graphenbasierten Ansatz beendet werden. Im Kapitel 7 wird die Darstellung dieses Ansatzes fortgeführt, und es wird untersucht, wie man unter den gegebenen Voraussetzungen globale Verklemmungssituationen behandeln kann.

b) Die 2PC-Agent-Methode

Die *2PC-Agent-Methode* (2PCA-Methode) wird in [WV90, VW92] vorgestellt und ist eng mit den bisher in diesem Abschnitt vorgestellten Ansätzen verbunden, da auch bei diesem Ansatz die Interaktionen von lokalen und globalen Transaktionen durch die Restriktion des Zugriffs auf bestimmte Mengen von Datenobjekten beschränkt werden. Im Gegensatz zu den bisherigen Ansätzen wiederholt jedoch diese Methode im Falle einer Ausnahmesituation während der "commit"-Prozedur nicht nur die "write"-Operationen, sondern alle Operationen der abgebrochenen Subtransaktion. Zunächst sollen jedoch erst einmal die grundsätzlichen Systembedingungen für diesen Ansatz geklärt werden.

Die bei der 2PC-Agent-Methode verwendete Architektur unterscheidet sich in einigen Details von der bisher verwendeten Referenzarchitektur. Abbildung 6.3 stellt die hier geltende Architektur dar. Die Funktionalität des GTM wird in diesem Ansatz verteilt. Der verteilte Transaktionsmanager (*DTM – "distributed transaction manager"*) besteht aus einer Menge Koordinatoren (*"coordinators"*), die in denjenigen Knoten plaziert sind, die globale Transaktionen ausführen möchten (*"coordinating sites"*), und einer Menge von Servern, die hier *"2PC-Agents"* (2PCA) genannt werden und in den partizipierenden Knoten (*"participating sites"*) angesiedelt sind und dort mit den lokalen Transaktionsverwaltungen (LTM – *"local transaction managers"*) verbunden sind. Zwischen einem Koordinator und den in die Verarbeitung einer globalen Transaktion involvierten Servern wird zur Terminierung der globalen Transaktion ein 2PC-Protokoll in der bekannten Art und Weise gefahren.

Eine globale Transaktion wird über das globale Interface (GI) zur Verarbeitung an den Koordinator übergeben. Der Koordinator erzeugt daraus eine Menge *globaler Subtransaktionen* und reicht diese operationsweise an die entsprechenden 2PCAs weiter und gibt gleichzeitig die Ergebnisse an die globale Applikation zurück. Eine globale Subtransaktion besteht dabei aus den üblichen Manipulationskommandos für Datenbanken, z.B. SQL-Kommandos. Für jede globale Transaktion wird maximal eine globale Subtransaktion pro Datenbankknoten erzeugt. Wenn den Koordinator die globale "commit"-Operation erreicht, wird das "2PC-Protokoll" gestartet. Der 2PCA bildet aus der globalen Subtransaktion eine oder mehrere *lokale Subtransaktionen*, die über das lokale Interface (LI)

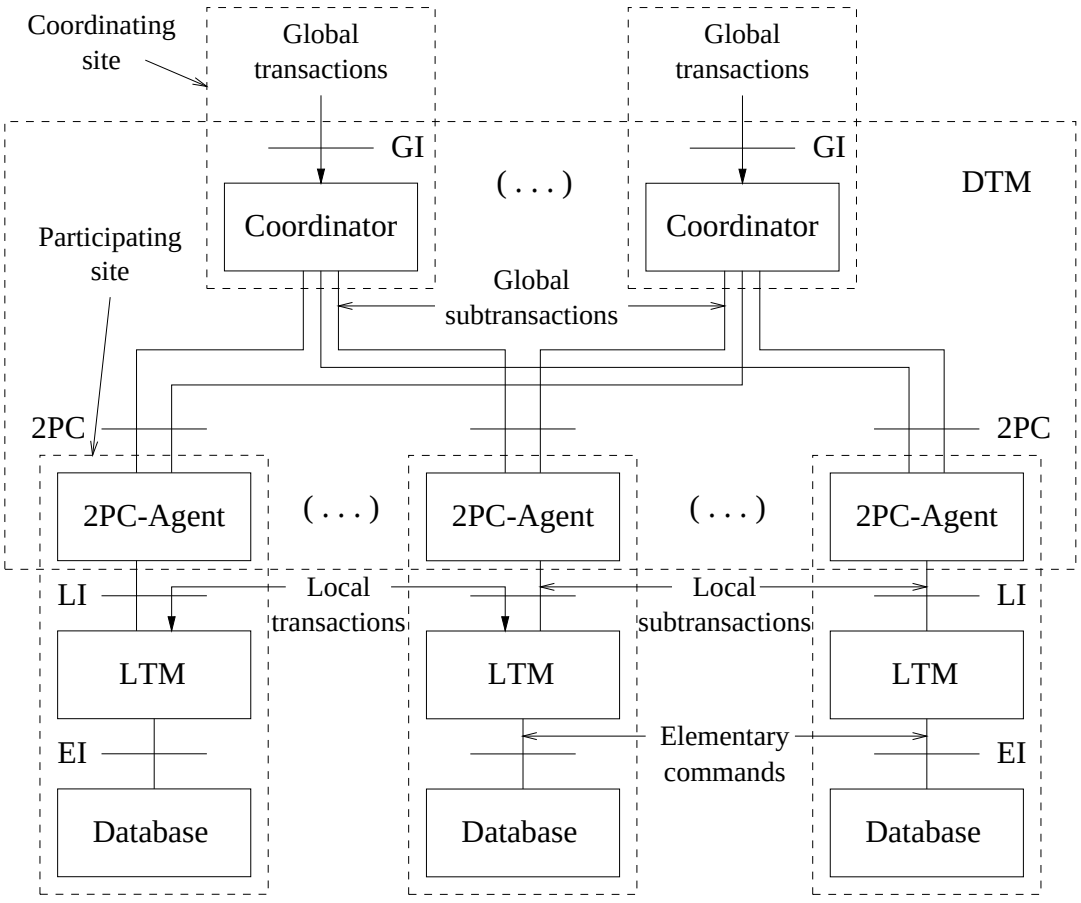

Abbildung 6.3: 2PCA-Architekturmodell

an den LTM zur Verarbeitung weitergereicht werden. Eine lokale Subtransaktion pro globale Subtransaktion ist dabei der Normalfall. Kommt es jedoch zu einem unilateralen Abbruch der lokalen Subtransaktion oder einem Knotenfehler, während die globale Subtransaktion im "prepared"-Zustand[3] ist, und der Koordinator entscheidet sich für ein "commit" der globalen Transaktion, so muß der 2PCA die lokale Subtransaktion wiederholen, so daß eine globale Subtransaktion aus mehreren lokalen Subtransaktionen bestehen kann.

[3]Da [WV90, VW92] in ihren Ausführungen eine Unterscheidung zwischen lokalen und globalen Subtransaktionen vornehmen, erscheint die Übernahme der ursprünglichen 2PC-Nachrichten und -Zustände für globale Subtransaktionen durchaus praktikabel und sinnvoll. Die 2PCAs sichern für globale Subtransaktionen durchaus ähnliche Eigenschaften wie 2PC-taugliche DBMSe: globale Subtransaktionen in einem "prepared"-Zustand können nicht durch das CDBMS abgebrochen werden. Für die zugehörigen lokalen Subtransaktionen trifft dies jedoch nicht zu. Das heißt, wenn eine globale Subtransaktion in einem "prepared"-Zustand ist, ist die entsprechende lokale Subtransaktion in einem "ready"-Zustand.

Folgende Annahmen und Restriktionen gelten nun für die 2PCA-Methode und die beteiligten Systeme:

- **DDF** ("deterministic decomposition function"): Im LTM gibt es eine zeitunabhängig deterministische Dekompositionsfunktion, die jedes höhersprachige Datenbankmanipulationskommando in eine Sequenz elementarer r/w-Operationen transformiert.
- **RR** ("rollback recovery"): Wird im lokalen Datenbanksystem eine Transaktion abgebrochen, so stellt der LTM für alle modifizierten Datenobjekte den entsprechenden "before image"-Zustand wieder her.
- **RTT** ("real time transparency"): Zwei identische Folgen von Datenbankmanipulationskommandos liefern unabhängig vom Ausführungszeitpunkt die gleichen Ergebnisse, wenn die gelesenen Datenobjekte die gleichen Werte ausweisen.
- **SRS** ("serializable and rigoros histories"): Alle lokalen Schedules sind serialisierbar und rigoros. Die LTM können dies durch den Einsatz von S2PL- oder gleichwertiger Verfahren realisieren.
- **TW** ("trustworthiness"): Nach einer bestimmten Anzahl von Wiederholungen einer lokalen Subtransaktion wird jede globale Transaktion, die mit "commit" beendet werden soll, auch so enden.
- **UAN** ("unilateral abort notification"): Der 2PCA ist über jeden unilateralen Abbruch, der sich ereignet hat, informiert.
- **DLU** ("denied local updates"): Lokale Transaktionen dürfen keine Datenobjekte modifizieren, die an eine globale Transaktion gebunden sind.

Die DLU-Forderung stellt eine der wesentlichen Restriktionen dieses Systems dar. Datenobjekte sind an eine globale Transaktion gebunden, wenn sie von ihr gelesen oder geschrieben wurden und die globale Transaktion noch nicht beendet oder abgebrochen wurde. Folglich darf kein von einer globalen Subtransaktion gelesener oder geschiebener Wert geändert werden, während diese aktiv oder im "prepared"-Zustand ist. Demnach muß verhindert werden, daß lokale Transaktionen nach dem Abbruch einer lokalen Subtransaktion, die wiederholt werden muß, vor der Wiederholung deren gelesene oder geschriebene Werte modifizieren. Auch wenn die DLU-Anforderung auf den Zeitraum der Ausführung einer globalen Transaktion beschränkt ist, ist deren Durchsetzung in den autonomen lokalen DBMSen aus meiner Sicht nur möglich, wenn die r/w-Operationen lokaler Subtransaktionen auf globale Datenobjekte beschränkt bleiben, während lokale Transaktionen zwar globale Datenobjekte lesen, nicht jedoch modifizieren dürfen.

Die Zielsetzung der 2PCA-Methode ist die Sicherung der globalen Serialisierbarkeit von Ausführungen in Umgebungen, in denen unilaterale Transaktionsabbrüche und lokale Knotenfehler erlaubt sind. Die 2PCA-Methode stellt dabei die Viewserialisierbarkeit

des globalen Schedules sicher, indem sie gewährleistet, daß sich durch die möglichen Ausnahmesituationen keine Änderung der Sichten lokaler und globaler Transaktionen ergibt. In einer vereinfachten Version wird die Quasiserialisierbarkeit globaler Abarbeitungen gewährleistet.

Der grundlegende Recovery-Ansatz ist dabei, wie bereits erwähnt, die vollständige Wiederholung einer lokalen Subtransaktion, falls für eine globale Subtransaktion, die sich im "prepared"-Zustand befindet und mit "commit" beendet werden soll, die zugehörige lokale Subtransaktion abgebrochen wurde. Die Wiederholung erzeugt eine neue, eigenständige lokale Subtransaktion. Zu diesem Zweck verwaltet der 2PCA, wie in vorangegangenen Ansätzen eine Log-Datei, die hier *"agent log"* genannt wird.

Im Falle von fehler- und abbruchfreien lokalen Ausführungen ist durch die Rigorosität der lokalen Schedules und die Anwendung eines 2PC-basierten Ansatzes die Serialisierbarkeit der globalen Schedules gesichert. Problematisch sind Situationen, in denen durch lokale Knotenfehler oder unilaterale Abbrüche in den CDBMSen Sperren für Datenobjekte freigegeben werden, die an globale Subtransaktionen im "prepared"-Zustand gebunden sind. Im Beispiel 3.3 wurde gezeigt, wie durch die Ausführung lokaler Transaktionen zwischen Abbruch und Wiederholung der lokalen Subtransaktion, die auf Datenobjekte der Subtransaktion zugreifen, die globale Serialisierbarkeit verletzt werden kann. Mit der 2PCA-Methode wird nun ein optimistischer Ansatz vorgestellt, der das dargestellte Problem lösen soll.

Zunächst werden durch den 2PCA alle Operationen einer globalen Subtransaktion an den entsprechenden LTM weitergereicht. Der 2PCA verläßt sich dabei voll auf die lokalen Serialisierungsmechanismen und führt selbst keine synchronisierenden Aktivitäten durch. Erreicht den 2PCA eine "prepare"-Nachricht, wird durch ihn eine *"prepare"-Zertifikation* ("prepare certification") durchgeführt, die überprüft, ob trotz möglicherweise nachfolgend auftretender Ausnahmesituationen jederzeit das spätere "commitment" gesichert werden kann. In diesem Schritt werden dabei Interaktionen mit anderen globalen Subtransaktionen untersucht, d.h. es wird konkret getestet, ob sich durch eine mögliche Wiederholung der lokalen Subtransaktion keine Änderung der Sicht der zu zertifizierenden globalen Subtransaktion ergeben kann, ungeachtet möglicher lokaler Transaktionen. Die "prepare"-Zertifikation allein sichert die Quasiserialisierbarkeit der globalen Abarbeitung und bildet somit einen "Ein-Schritt-Mechanismus" ("single step certifier") zur Sicherung einer globalen Scheduleeigenschaft. War die "prepare"-Zertifikation erfolgreich, kann die globale Subtransaktion den "prepared"-Status einnehmen und die entsprechende Nachricht an den Koordinator senden.

Nach dem Erhalt der "commit"-Nachricht wird als zweiter Schritt ("two step certifier") eine "commit"-Zertifikation durchgeführt, die überprüft, ob durch lokale Transaktionen in Fehler- oder Abbruchfällen keine Zyklen im globalen Serialisierungsgraphen entstehen. Die beiden Zertifizierungsschritte sollen nun detailliert beschrieben werden:

"prepare"-Zertifikation:

Die "prepare"-Zertifikation sichert, daß sich durch die Wiederholung einer lokalen Subtransaktion die Sicht der globalen Subtransaktion und damit der gesamten globalen Transaktion nicht ändert. Die folgenden Korrektheitsbedingungen ("2PCA correctness invariant") werden dabei durchgesetzt:

1. Keine globale Subtransaktion wird in den "prepared"-Zustand gebracht, wenn die entsprechende lokale Subtransaktion unilateral abgebrochen wurde.

2. Zwei globale Subtransaktionen, die im "prepared"-Zustand sind, haben keine elementaren, miteinander in einem Konflikt stehenden Operationen.

Es läßt sich leicht klar machen, daß die Realisierung dieser Bedingung ausreicht, um die Sicht und damit die Serialisierungsreihenfolge einer globalen Transaktion konstant zu halten. Da die DLU-Eigenschaft für lokale Schedules gilt, kann eine Änderung der Sicht nur durch eine lokale Subtransaktion erfolgen, deren globale Subtransaktion noch nicht den "prepared"-Zustand erreicht hat. Die zu wiederholende Subtransaktion wird aufgrund der SRS-Anforderung in der Ausführung so lange verzögert, bis die ändernde Subtransaktion endet oder abbricht. Da diese aber aufgrund der obigen Korrektheitsbedingung abbrechen muß und alle ihre Effekte zurückgesetzt werden (RR), bleibt die Sicht der globalen Transaktion unverändert.

Kann die obige Korrektheitsbedingung durch die 2PCA durchgesetzt werden, dann können die Subtransaktionen mit "commit" beendet werden, ohne die Serialisierbarkeit zwischen den globalen Transaktionen zu gefährden, unabhängig davon, wann eine "commit"-Operation beim 2PCA eintrifft. Basis für die Implementation der obigen Bedingungen bildet das Ausschließen von Konflikten zwischen allen globalen Subtransaktionen im "prepared"-Zustand und der zu zertifizierenden globalen Subtransaktion. Zwei globale Subtransaktionen stehen unter der Annahme rigoroser lokaler Schedules offensichtlich nicht miteinander in Konflikt (weder direkt noch indirekt), wenn beide ihre lokalen Datenbankoperationen beendet haben, aber noch nicht beendet oder abgebrochen wurden. Man sagt, die Subtransaktionen sind in einem *"alive"*-Zustand. Bei der Zertifikation einer globalen Subtransaktion muß der 2PCA demnach überprüfen, ob die zu testende Subtransaktion zur selben Zeit in einem "alive"-Zustand war, in dem alle momentan im "prepared"-Zustand verweilenden Subtransaktionen ebenfalls "alive" waren. Der 2PCA verwaltet zu diesem Zweck "alive time"-Intervalle für alle globalen Subtransaktionen in einem "prepared"-Zustand, die die Zeitspanne von der letzten Datenbankoperation bis zum aktuellen Zeitpunkt umfassen[4]. Demnach kann eine zu zertifizierende Subtransaktion den "prepared"-Zustand erreichen, wenn ihr "alive time"-Intervall einen nichtleeren Durchschnitt mit allen Subtransaktionen in einem "prepared"-Zustand hat.

[4]Wurde die lokale Subtransaktion einer globalen Subtransaktion im "prepared"-Zustand unilateral abgebrochen, so umfaßt das Intervall zunächst den Zeitraum von der letzten Datenbankoperation bis zum Abbruch (es gilt UAN). Bei der Wiederholung der lokalen Subtransaktion wird mit der letzten Datenbankoperation ein neuer Intervall gestartet.

Mit Hilfe dieses Ansatzes kann die 2PCA-Methode sicherstellen, daß die zukünftige Wiederholung einer lokalen Subtransaktion stets ausgeführt werden kann, ohne die globale Serialisierungsreihenfolge zu gefährden.

"commit"-Zertifikation:

Wenn beim 2PCA die "commit"-Operation einer globalen Subtransaktion eintrifft, sollte der Server die lokale Subtransaktion mit "commit" beenden. Jedoch besteht auch unter den bisher aufgeführten Bedingungen die Möglichkeit, daß ein solches "commit" Zyklen im globalen Serialisierbarkeitsgraphen erzeugt. Das Problem liegt darin, daß lokale Transaktionen durch unilaterale Abbrüche nichtserialisierbare Sichten erhalten haben können. Beispiel 6.1 stellt eine derartige Situation dar.

Offensichtlich besteht das Problem darin, daß in einer solchen Situation die Reihenfolge der "commit"-Operationen in den CDBSen unterschiedlich ist, d.h. die Forcierung einer einheitlichen "commit"-Reihenfolge in allen beteiligten Knoten stellt eine ausreichende Lösung dieses Problems dar. Es stellt sich jedoch die Frage, wie sich eine solche einheitliche "commit"-Reihenfolge ermitteln läßt, ohne durch ihre Umsetzung eine große Anzahl von globalen Transaktionsabbrüchen zu verursachen. Eine willkürlich festgelegte Reihenfolge kann den lokalen Serialisierungsreihenfolgen widersprechen und so viele Transaktionsabbrüche nach sich ziehen. Ebenso ist ein synchrones Senden der "commit"-Operationen zu den Servern, d.h. der Koordinator verzögert eine "commit"-Entscheidung so lange, bis die vollständige Ausführung der vorangegangenen "commit"-Operation von allen 2PCAs bestätigt wurde, wenig geeignet, da dies zu globalen Deadlocks führen kann, in die der Koordinator involviert ist [WV90].

In [VW92] wird ein Algorithmus vorgestellt, der von einer global bestimmten Reihenfolge ausgeht, die äquivalent zur einzigen Serialisierungsreihenfolge ist, falls eine solche existiert. Zu diesem Zweck bekommt jede Transaktion T_i eine Seriennummer $SN(T_i)$. Es wird gefordert, daß:

> *Wenn T_i in der lokalen Serialisierungsreihenfolge T_j vorangeht, dann gilt* $SN(T_i) < SN(T_j)$.

$SN(T_i)$ und $SN(T_j)$ müssen während der Ausführung bestimmt werden. Dies ist möglich, wenn alle ursprünglichen Datenbankkommandos ausgeführt wurden (die LTM haben die lokalen Subtransaktionen entsprechend der Konflikte abgearbeitet) und die Applikation sendet die "commit"-Operation an den Koordinator. Der Koordinator gibt in diesem Moment der globale Transaktion eine global einmalige Seriennummer und schickt diese zusammen mit dem "prepare"-Kommando an die involvierten 2PCAs. Diese speichern die Seriennummern der einzelnen globalen Transaktionen. Wenn die "commit"-Operationen später in den 2PCAs eintreffen, realisiert jeder Server eine "commit"-Reihenfolge entsprechend der gegebenen Ordnung der Seriennummern.

An dieser Stelle soll die Erläuterung der 2PCA-Methode abgeschlossen werden. In [WV90] wird ein ausführlicher Beweis der Korrektheit der 2PCA-Methode gegeben. Ein erweiterter Ansatz zur "prepare"-Zertifikation, vergleichende Betrachtungen zur

"commit"-Graphenmethode und diverse Algorithmen zur Zertifikation und für das "alive time"-Intervall-Management finden sich in [VW92].

Die hier vorgestellten Zertifikationsalgorithmen sind in das Prototypsystem HERMES des "Laboratory for Information Processing" des "Technical Research Centre of Finland (VTT)" implementiert. Dieses System integriert zwei kommerzielle Datenbankprodukte: SQL-Server (Sybase Inc.) und INGRES (Ask Computer Systems). Dabei wird bei dem Sybase-Produkt das vorhandene 2PC-Interface genutzt, während zur Verbindung mit dem INGRES DBMS ein 2PCA genutzt wird.

6.3 Der Retry-Ansatz

Der *Retry-Ansatz* ist ein weiterer alternativer Ansatz zur Sicherung der Atomarität und Dauerhaftigkeit globaler Transaktionen. Im Gegensatz zum vorherigen Ansatz sollen hier Situationen betrachtet werden, in denen globale Transaktionen lokale Datenobjekte lesen und schreiben dürfen. In solchen Fällen ist der Redo-Ansatz zur Behebung globaler Transaktionsfehler, d.h. von Situationen in denen Subtransaktionen einer globalen Transaktion in bestimmten Knoten erfolgreich enden und in anderen Knoten abbrechen, nicht anwendbar.

Nehmen wir an, daß eine globale Transaktion GT_i aus zwei Subtransaktionen in den Knoten s_1 und s_2 besteht, wobei bei einem globalen Transaktionsfehler die Subtransaktion ST_{i1} im Knoten s_1 mit "commit" abschließt, wogegen im Knoten s_2 die Subtransaktion ST_{i2} abbricht. Der Retry-Approach geht davon aus, daß in einem solchen Fall die komplette Subtransaktion vom GTM erneut gestartet wird, d.h. auf s_2 läuft als neue Transaktion die Subtransaktion ST'_{i2}, wobei ST'_{i2} dabei Werte lesen und schreiben kann, die von der ursprünglichen Verarbeitung abweichen können. Dies ist jedoch nur dann möglich, wenn der GTM Ausführungszustände der globalen Transaktion GT_i, z.B. lokale Variablen des Programms, das GT_i abgearbeitet hat, und die ST_{i2} benötigte, gespeichert hat. Außerdem dürfen die Originalwerte, die ST_{i2} gelesen hat, nicht an andere Subtransaktionen von T_i weitergereicht worden sein, da diese Werte bei der erneuten Abarbeitung der Subtransaktion bereits geändert und damit ungültig sein können. Folglich ist die NVD-Eigenschaft zwischen ST_{i2} und den anderen Subtransaktionen von T_i Voraussetzung für die Anwendung des Retry-Ansatzes im Knoten s_2.

Eine weitere wichtige Voraussetzung ist, daß die Subtransaktionen einer globalen Transaktion *wiederholbar* (*"retriable"*) sein müssen, d.h. es muß sichergestellt sein, daß eine Subtransaktion nach einer endlichen Anzahl von Versuchen auch mit "commit" abgeschlossen werden kann. Dies ist nicht zwingend der Fall, da in der Zeit zwischen ursprünglicher Ausführung und erster Wiederholung eine andere Transaktion die Zustände in der Datenbank so geändert haben kann, daß durch die Wiederholung Konsistenzbedingungen verletzt werden und so die Subtransaktion wiederholt abbricht.

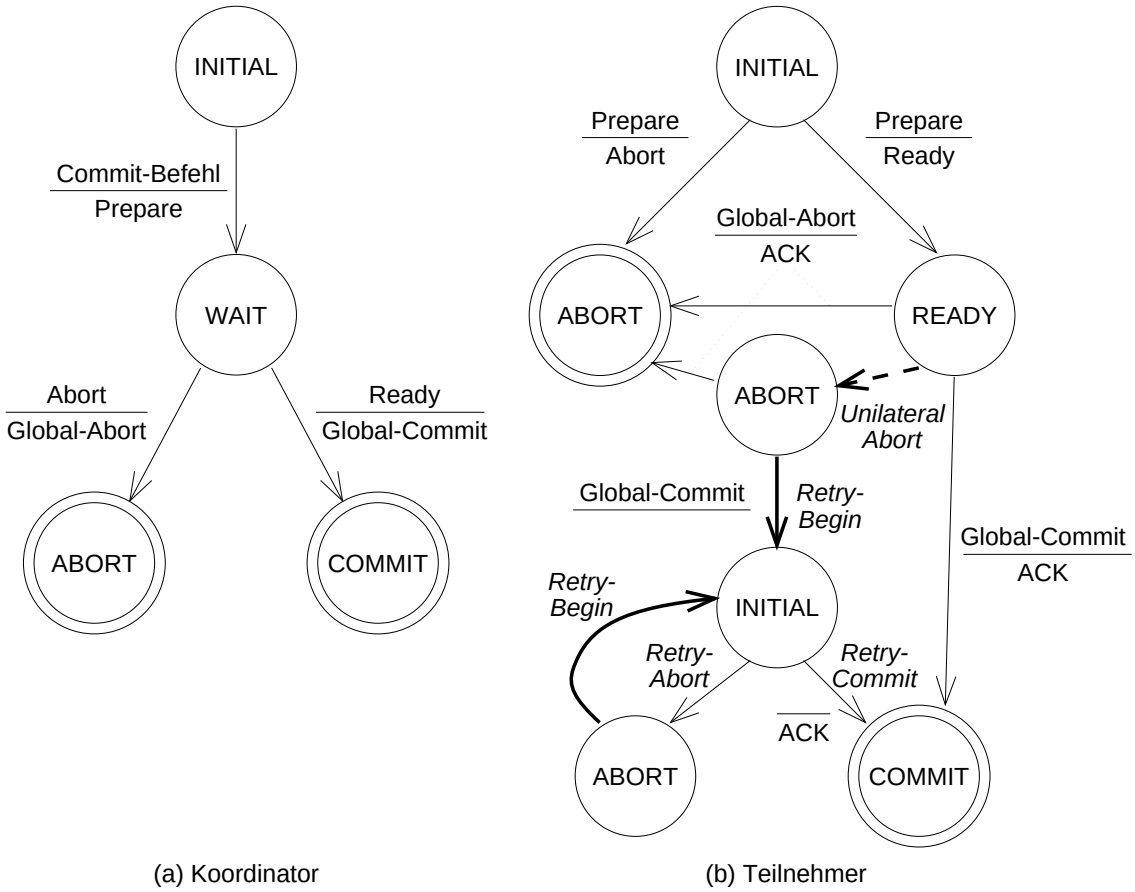

Abbildung 6.4: Zustandsübergänge beim Retry-Ansatz

Die Arbeitsweise des Retry-Ansatzes wird durch das Zustandsübergangsdiagramm in Abbildung 6.4 dargestellt. Der GTM fungiert als Koordinator, die lokalen DBMSe inklusive der zugehörigen Server bilden die Teilnehmer. Die Endzustände globaler bzw. lokaler Transaktionen sind durch Doppelkreise dargestellt. Die Arbeitsweise des GTM entspricht der im 2PC-Protokoll. Der GTM startet die Terminierung einer globalen Transaktion mit einer "prepare"-Nachricht an die lokalen Server. Konnten in einem Knoten alle Operationen der entsprechenden Subtransaktion abgearbeitet werden, gibt der Server eine "ready"-Nachricht an den Koordinator, ansonsten wird ein "abort" gemeldet. Melden alle involvierten Server ein "ready" der einzelnen Subtransaktionen, entscheidet sich der GTM für ein globales "commit" der Transaktion, ist dies nicht der Fall, wird die globale Transaktion abgebrochen. Die globalen Entscheidungen werden an die Server weitergereicht, die für die lokale Umsetzung verantwortlich sind. Da der "ready"-Zustand im Gegensatz zum "prepared"-Zustand nicht vom lokalen DBMS zur Verfügung gestellt wird, kann eine Subtransaktion in diesem Zustand vor dem globalen "commit" durch die lokale Transaktionsverwaltung abgebrochen werden. Ist dies der Fall und hat der GTM auf "global commit" entschieden, muß die Transaktion wiederholt werden, und zwar so lange, bis die Subtransaktion erfolgreich endet.

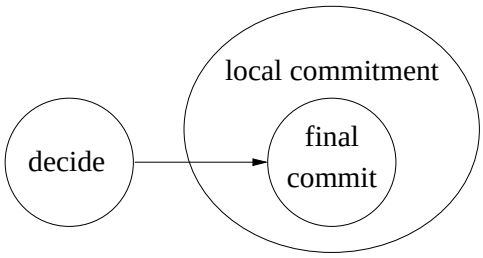

Abbildung 6.5: Lokales "commitment" *nach* der globalen Entscheidungsfindung

In [MR91] wird diese Art des lokalen Abschlusses der Subtransaktionen als Beendigung *nach* der globalen Entscheidung bezeichnet, da die lokalen "commit"-Prozeduren erst starten, nachdem der GTM seine Terminierungsentscheidung getroffen und sie an die betroffenen Server weitergereicht hat. Abbildung 6.5 verdeutlicht diese Art des "commitments".

Nachfolgend soll untersucht werden, inwieweit der Retry-Ansatz mit den Synchronisationsmechanismen aus den Kapiteln 4 und 5 kombiniert werden kann. Keine zusätzlichen Bedingungen sind erforderlich, wenn die lokale Serialisierbarkeit (LSR) als Kriterium korrekter Ausführungen herangezogen wird, da nach dem erfolgreichen Ende einer wiederholten Transaktion die lokalen Schedules weiterhin serialisierbar sind. Wird dagegen mit der zweistufigen Serialisierbarkeit gearbeitet, ist es nötig, eines der Protokolle aus dem vorangegangenen Abschnitt zu verwenden, die sicherten, daß die Projektion des globalen Schedules auf die Operationen globaler Transaktionen serialisierbar ist. Dabei muß der GTM eine wiederholte Transaktion als Teil der ursprünglichen globalen Transaktion ansehen. Verwendet man globale Serialisierbarkeitskriterien im Zusammenhang mit den Retry-Ansatz, gelten die Aussagen des Redo-Ansatzes. Durch die wiederholte Ausführung können neue indirekte Konflikte zwischen globalen Transaktionen entstehen, so daß eine der im Abschnitt 6.2 vorgestellten Techniken zum Einsatz kommen muß, um dadurch eventuell entstehende Zyklen zu verhindern.

6.4 Der Compensate-Ansatz

Der letzte, in diesem Kapitel vorgestellte allgemeine Ansatz zur Behandlung lokaler Fehler- und Abbruchsituationen ist der *Compensate-Ansatz*. Betrachten wir wiederum die Situation, in der eine globale Transaktion in einem Datenbankknoten mit "commit" beendet und in einem anderen Knoten abgebrochen wird. Im Gegensatz zum vorangegangenen Retry-Ansatz, in dem versucht wurde, die abgebrochene Subtransaktion zu wiederholen, versucht der Compensate-Ansatz, die mit "commit" beendete Subtransaktion zu kompensieren, d.h. es wird versucht, die Effekte der beendeten Transaktion zurückzusetzen und so das Ergebnis einer atomar abgebrochenen Transaktion zu erzielen. Dies kann man erreichen, indem man für eine mit "commit" beendete Subtransaktion ST_i^k einer globalen Transaktion GT_i im Knoten s_k eine spezifische Kompensationstrans-

aktion CT_i^k in s_k ausführt, die aus semantischer Sicht alle Effekte von ST_i^k zurücksetzt. Da jedoch die Effekte von ST_i^k zumindest den lokalen Transaktionen sichtbar gewesen sind, ist das Ergebnis nicht identisch zu dem Zustand, der ohne die Ausführung von ST_i^k in der Datenbank bestehen würde, sondern nur zu diesem Zustand semantisch äquivalent.

Die Kompensationstransaktion einer globalen Subtransaktion ist selbst eine reguläre Transaktion und muß als solche durch ihre Ausführung die Konsistenz der Datenbank erhalten. Dabei kann die Kompensationstransaktion nicht nur aus den inversen Funktionen der rückzusetzenden Transaktion bestehen, sondern auch verschiedene andere Aktionen enthalten, ohne die geforderte Konsistenzwahrung zu verletzen. Es ist ebenfalls denkbar, daß Effekte der ursprünglichen Transaktion sich auf andere Datenbankknoten ausdehnen, z.B. durch eine weitere globale Transaktion, die einen modifizierten Wert liest und Replikate in verschiedenen Knoten davon anlegt. Soll in diesem Fall die ursprüngliche Transaktion kompensiert werden, muß die Kompensationstransaktion auch die Effekte in den anderen Knoten zurücksetzen. Es ist natürlich nicht praktikabel, daß die Kompensationstransaktion einer globalen Subtransaktion auf verschiedene Knoten zugreifen muß. Aus diesem Grund werden bei den Kompensationsaktionen bestimmte Restriktionen vorgenommen, die unerwünschte Ausweitungen einer Kompensationstransaktion auf andere Knoten verhindern:

1. Keine andere globale Transaktion darf die geschriebenen Effekte einer globalen Subtransaktion sehen, bevor die Kompensationstransaktion nicht vollständig abgearbeitet wurde.

2. Globale Transaktionen dürfen keine wertemäßigen Abhängigkeiten zwischen ihren Subtransaktionen haben.

Nachfolgend wird davon ausgegangen, daß beide Bedingungen gelten und so die Kompensationstransaktionen auf die Knoten der zurückzusetzenden Subtransaktionen beschränkt bleiben. Wie bereits erwähnt, führt die Abarbeitung von Kompensationstransaktionen nicht zu der ursprünglich geforderten "Standardatomarität" von Transaktionen, sondern führt zu einem eingeschränkten Atomaritätsansatz, der als semantische Atomarität bezeichnet wird:

Definition 6.2 (Semantische Atomarität) Es sei GT_i eine globale Transaktion. Weiter sei $\mathcal{CT}_i$ die Menge der lokalen Kompensationstransaktion $CT_i^1, \ldots, CT_i^k$ der Subtransaktionen von GT_i, eine für jeden Knoten, auf dem GT_i arbeitet. Die globale Transaktion GT_i ist genau dann *semantisch atomar*, wenn GT_i entweder in allen Knoten, in denen GT_i ausgeführt wurde, mit "commit" beendet wurde oder wenn CT_i^j in allen Knoten mit "commit" beendet wurde, in denen auch GT_i mit "commit" endete. □

Da der Transaktionsbegriff in der Regel die "vollständige" Atomarität einer Ausführung impliziert, werden in der Literatur für Transaktionsansätze mit semantischer Atomarität oftmals alternative Transaktionsbegriffe, wie z.B. der Begriff "Saga", verwendet. Nachfolgend sollen einige Ansätze, die semantische Atomarität garantieren, kurz erläutert werden.

a) Das O2PC-Protokoll

Das *optimistische Zwei-Phasen-Sperrprotokoll* (O2PC-Protokoll) wird in [LKS91b] als Ansatz zur Realisierung semantischer Atomarität vorgestellt. Das O2PC-Protokoll stellt dabei eine leicht modifizierte Version des *verteilten 2PL* ("distributed 2PL") aus [BHG87] dar.

Das verteilte 2PL-Protokoll sichert durch eine Kombination von lokalen 2PL-Schedulern mit einem 2PC-Protokoll globale Serialisierbarkeit. Es wird davon ausgegangen, daß der GTM die Operationen einer globalen Transaktion erst dann zur Verarbeitung an die lokalen Systeme weiterreicht, wenn die Ausführung der jeweils vorangegangenen Operation bestätigt wurde. Folglich hat eine globale Transaktion zum Zeitpunkt der "prepare-to-commit"-Operation alle benötigten lokalen Sperren erhalten. Mit dem Eintreffen der "prepare-to-commit"-Operation in den lokalen Systemen kann die Entsperrphase der jeweiligen Subtransaktion beginnen. Zur Vermeidung kaskadierender Abbrüche und der Anwendung zustandsbasierter Recovery-Mechanismen ist es nötig, exklusive (Schreib-) Sperren bis zum Eintreffen der globalen Entscheidung zu halten. Ein S2PL-Scheduler sichert diese Eigenschaft. Geteilte (Lese-) Sperren können jedoch mit dem Eintreffen der "prepare-to-commit"-Operation freigegeben werden.

Das Halten der Schreibsperren bis zum "commit"-Zeitpunkt verursacht Verzögerungen im Ablauf anderer Transaktionen und ist eigentlich nur nötig, wenn die sperrende Transaktion aus globaler Sicht abgebrochen werden soll. Das O2PC-Protokoll geht dagegen von der optimistischen Annahme aus, daß in den meisten Fällen die Transaktion erfolgreich beendet wird und so die Sperren schon vorzeitig freigegeben werden könnten. Da eine "prepare-to-commit"-Operation erst zur Ausführung gebracht wird, wenn die globale Transaktion alle lokalen Sperren erhalten hat und alle Operationen ausgeführt wurden, sind Fehlersituationen im weiteren Verlauf recht unwahrscheinlich.

Unter dieser Annahme arbeitet das O2PC-Protokoll wie folgt: Die Verarbeitung der Datenbankoperationen einer globalen Transaktion erfolgt analog zum verteilten 2PL-Verfahren. Trifft jedoch eine "prepare-to-commit"-Operation in einem Server ein, versucht dieser unter der optimistischen Annahme die Subtransaktion lokal zu beenden. Das Ergebnis wird dem GTM mitgeteilt und alle lokalen Sperren der Subtransaktionen werden freigegeben. Konnten alle Subtransaktionen einer globalen Transaktion mit "commit" beendet werden, wird die globale Transaktion vom GTM als "committed" deklariert, ansonsten muß sie als "aborted" deklariert werden. Im letzteren Fall ist es nötig, lokale Effekte von erfolgreich beendeten Subtransaktionen der globalen Transaktion zurückzusetzen. Der Einsatz von Kompensationstransaktionen ist folglich im O2PC-Protokoll nötig, um die semantische Atomarität einer globalen Transaktion sicherzustellen. Abbildung 6.6 stellt die möglichen Zustandsübergänge dieses Protokolls dar.

In dem allgemein angenommenen Fall einer erfolgreichen Abarbeitung globaler Transaktionen beendet das O2PC-Protokoll globale Subtransaktionen früher als im Standard-2PC-Protokoll und erhöhen somit die Parallelität globaler Transaktionen und den allgemeinen Transaktionsdurchsatz. [MR91] charakterisieren diesen Ansatz als ein "commitment" *vor* einer globalen Terminierungsentscheidung. Abbildung 6.7 bringt diesen

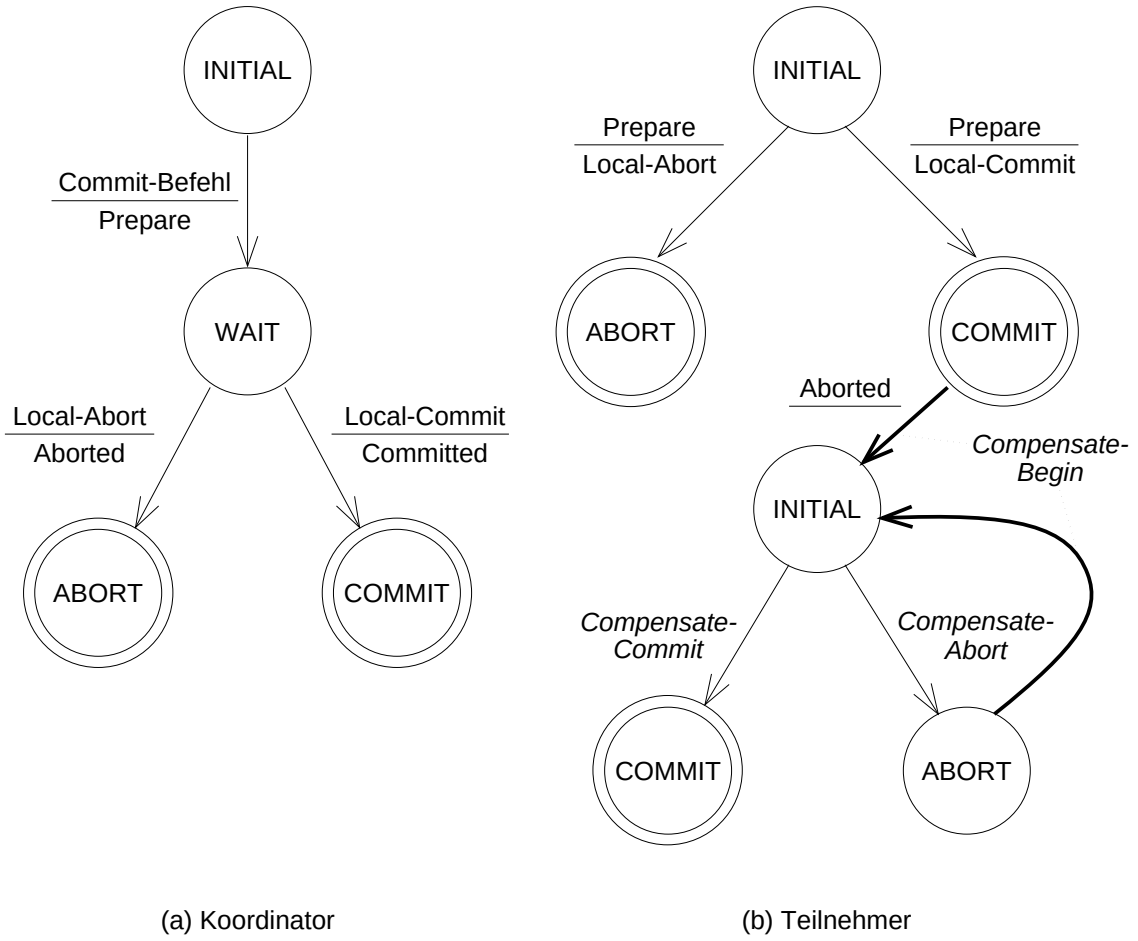

Abbildung 6.6: Zustandsübergänge beim O2PC-Protokoll

Aspekt zum Ausdruck.

[LKS91b] gehen in ihrem O2PC-Protokoll davon aus, daß alle teilnehmenden Knoten S2PL-Scheduler verwenden. Für die Kompensationstransaktion CT_i einer Transaktion T_i fordern sie, daß CT_i alle Effekte von T_i semantisch zurücksetzt, ohne kaskadierende Abbrüche von Transaktionen zu verursachen, die von T_i gelesen haben. Folglich garantiert die Ausführung von CT_i nicht, daß alle direkten und indirekten physikalischen Effekte von T_i zurückgesetzt werden, sondern nur, daß ein konsistenter Zustand basierend auf semantischen Informationen erreicht wird. Es liegt in der Natur der Kompensation, daß CT_i stets nach T_i serialisiert wird, jedoch muß zur Sicherung der semantischen Atomarität CT_i nicht notwendigerweise direkt nach T_i serialisiert werden. Eine wesentliche von [LKS91b] geforderte Eigenschaft der Kompensation ist deren "beharrliche" Ausführung, d.h. wenn einmal die Kompensation einer globalen Transaktion begonnen hat, muß garantiert werden, daß sie auch erfolgreich endet. Diese wie auch einige andere in [LKS91b] aufgestellte Forderungen impliziert, daß die Kompensationstransaktionen eher als lokale Transaktionen angesehen werden denn als Teile einer "globalen Kompensationstransak-

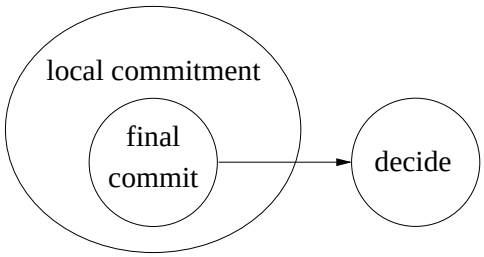

Abbildung 6.7: Lokales "commitment" *vor* der globalen Entscheidungsfindung

tion". Alle von einer lokal ausgeführten Kompensationstransaktion gehaltenen Sperren werden sofort nach ihrer Beendigung freigegeben und es wird kein atomares "commit"-Protokoll für die einheitliche Terminierung einer Menge semantisch zusammengehörender Kompensationstransaktionen eingesetzt. Dies ist einer der Hauptgründe dafür, daß mit diesem Ansatz keine globale Serialisierbarkeit, sondern nur "semantische Serialisierbarkeit" gewährleistet werden kann.

Eine wesentliche Anforderung für die Korrektheit dieses Ansatzes ist, daß eine globale Transaktion nur einen Zustand in der Datenbank liest, der aus globaler Sicht auch konsistent ist, d.h. es darf nicht passieren, daß eine globale Transaktion GT_i in einem Knoten s_k von einer mit "commit" beendeten Subtransaktion einer globalen Transaktion GT_j liest, während in einem anderen Knoten s_l die Subtransaktion von GT_i bereits die kompensierten Effekte von GT_j vorfindet. Die *Isolation der Rücksetzbarkeit* (IR – "isolation of recovery") [LKS91a] bezeichnet eine Eigenschaft, die garantiert, daß eine globale Transaktion nicht gleichzeitig die Effekte erfolgreich abgearbeiteter und abgebrochener bzw. kompensierter Subtransaktionen einer globalen Transaktion sieht. Verbietet man, daß globale Subtransaktionen zwischen dem Ende einer zu kompensierenden Subtransaktion und der entsprechenden Kompensationstransaktion serialisiert werden, ist die IR-Eigenschaft trivialerweise erfüllt.

In [LKS91b] werden zunächst die Bedingungen analysiert, die zur Verletzung der IR-Eigenschaft führen. [LKS91b] untersuchen dazu den globalen Serialisierbarkeitgraphen auf sogenannte *reguläre Zyklen* ("regular cycles"), die die Verletzung der IR-Eigenschaft anzeigen und mindestens aus einer globalen und einer Kompensationstransaktion bestehen. Ein solcher regulärer Zyklus zwischen einer globalen Transaktion GT_2 und der Kompensationstransaktion CT_1 ($GT_2 \rightarrow CT_1 \rightarrow GT_2$) entsteht zum Beispiel, wenn im Knoten s_1 die Abhängigkeit $GT_1 \rightarrow GT_2 \rightarrow CT_1$ gilt, während in s_2 in der Reihenfolge $GT_1 \rightarrow CT_1 \rightarrow GT_2$[5] serialisiert wird. [LKS91b] stellen allgemeine Bedingungen auf, die immer erfüllt sind, wenn ein regulärer Zyklus auftritt und formulieren daraus Kriterien, die stets zu einer Verletzung der Bedingungen und damit zur Vermeidung regulärer Zyklen führen. Die Erfüllung dieser Kriterien ist somit ausreichend für die Realisierung der IR-Eigenschaft.

[5] In diesem Fall kann CT_1 auch das Standard-Recovery einer unilateral abgebrochenen Subtransaktion GT_1 sein, da [LKS91b] diesen lokalen Mechanismus ebenfalls als Kompensationstransaktion betrachten.

Diese Kriterien sind in einen Algorithmus P1 integriert wurden, der das O2PC-Protokoll erweitert und auf der Markierung von Knoten basiert. Jeder Knoten kann dabei im Hinblick auf eine bestimmte Transaktion GT_i markiert oder unmarkiert sein. Markierte Knoten können im Hinblick auf eine bestimmte Transaktion als "lokal beendet" oder "zurückgesetzt" markiert sein. Der Wechsel der Markierungen ist an den Nachrichtenfluß des O2PC gebunden, es ist somit kein zusätzlicher Nachrichtenaustausch nötig. Zunächst ist jeder Knoten unmarkiert im Hinblick auf eine Transaktion GT_i. Votiert ein Knoten im O2PC-Protokoll für "commit", so wird er als "lokal beendet" markiert, stimmt er für "abort", wird er mit der "zurückgesetzt"-Markierung versehen. Eine globale "commit"-Entscheidung hebt alle "lokal beendet"-Markierungen auf, eine globale "abort"-Benachrichtigung läßt "lokal beendet"-Markierungen zu "zurückgesetzt"-Markierungen werden. Die "zurückgesetzt"-Markierung eines Knotens im Hinblick auf eine bestimmte Transaktion wird nach dem Eintreten einer entsprechenden Bedingungen zurückgenommen. Das Protokoll P1 verhindert nun Situationen, in denen eine globale Transaktion auf einen Knoten zugreift, der für eine bestimmte andere Transaktion als "lokal beendet" markiert ist, während ein anderer Knoten, auf den ebenfalls zugegriffen werden soll, für die andere Transaktion eine "zurückgesetzt"-Markierung hat. Auf eine detailliertere Beschreibung des P1-Algorithmus, seiner Bedingungen und Strukturen soll an dieser Stelle verzichtet werden, da die grundlegende Idee des Algorithmus für das Verständnis dieses Ansatzes ausreichend ist.

Zusammenfassend läßt sich feststellen, daß das O2PC-Protokoll die Schwierigkeiten bei der Durchsetzung von Standardatomarität globaler Transaktionen in verteilten Systemen durch die Realisierung der schwächeren semantischen Atomarität vermeidet. Durch die Lockerung der Atomaritätsforderung kann die Serialisierbarkeit verloren gehen, jedoch nur wenn unilaterale Abbrüche oder lokale Fehlersituationen in der Abarbeitung auftreten.

b) ONT/MLT-Ansätze

Eine große Anzahl von Publikationen zum Transaktionsmanagement in verteilten, autonomen und heterogenen Datenbankumgebungen beschäftigt sich mit dem Einsatz erweiterter Transaktionsmodelle, beispielsweise *offen geschachtelter Transaktionen* (ONT – *"open nested transactions"*) oder *mehrschichtiger Transaktionen* (MLT – *"multi-level transactions"*) (z.B. [SWS91, WDSS93, SS93, DSW94, SSW95, MR91, MVN93]). Der Einsatz der erweiterten Transaktionsmodelle wird in der Regel mit den vielen Gemeinsamkeiten zur Transaktionsverwaltung in föderierten oder Multidatenbankumgebungen begründet. So findet jeweils ein Transaktionsmanagement auf mehreren Ebenen statt, der lokalen und globalen Ebene beim allgemeinen föderierten Ansatz und unterschiedlichen Abstraktionsebenen beim ONT/MLT-Management. Jedoch müssen sowohl der allgemeinere ONT-Ansatz als auch der speziellere MLT-Ansatz entsprechend modifiziert werden, um alle Anforderungen einer föderierten Transaktionsverwaltung zu erfüllen. Da bei den MLT- bzw. ONT-Ansätzen auch Kompensationstechniken angewendet werden, soll an dieser Stelle ein kurzer Überblick über diese Ansätze gegeben werden.

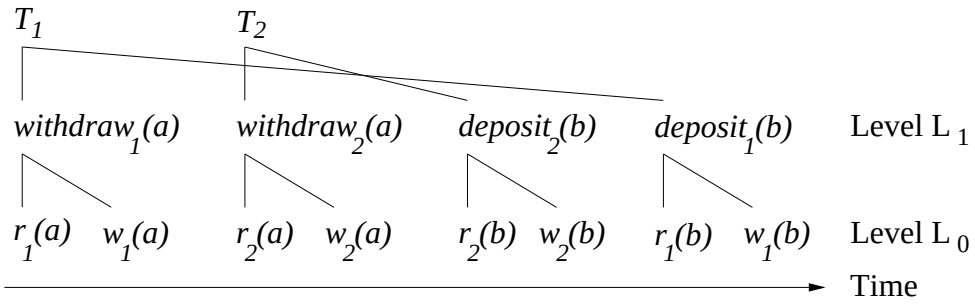

Abbildung 6.8: Beispiel zweier konkurierender MLT

Zunächst soll kurz das allgemeine MLT-Modell erläutert werden. Die MLT sind ein Spezialfall der offen geschachtelten Transaktionen, die als mehrschichtige Transaktionsbäume dargestellt werden und bei denen die Knoten innerhalb des Transaktionsbaums die Ausführung von Operationen auf unterschiedlichen Abstraktionsebenen in einem geschichteten System darstellen. Die Kanten im Transaktionsbaum repräsentieren die Implementation einer Operation, die in der Schicht L_{i+1} aufgerufen wurde, durch eine Folge von Operationsausführungen in der nächstniedrigeren Ebene L_i $(i = 0, \ldots, n)$. Alle MLT haben dabei Transaktionsbäume der gleichen Höhe, d.h. die Anzahl der Abstraktionsebenen ist für alle MLT gleich und hängt von der zugrundeliegenden Systemarchitektur ab. Das wesentliche Konzept bei der Synchronisation von MLT bilden die *ebenenspezifischen Konfliktrelationen* CON_i, die die Kommutativität bzw. Kompatibilität von Operationen reflektieren. Die Konflikte werden dabei mit Blick auf eine bestimmte Abstraktionsebene spezifiziert, nicht mit Blick auf deren physische Ausführung. Zwei L_i-Operationen f und g sind genau dann kommutativ (und damit konfliktfrei), wenn sie in jedem möglichen Systemzustand σ die beiden folgenden Bedingungen erfüllen:

1. Die Zustände, die aus den sequentiellen Ausführungen f, g und g, f im Zustand σ folgen, sind für andere L_i-Operationen nicht unterscheidbar.

2. Die Operationen f und g haben in beiden Ausführungsreihenfolgen die gleichen Rückgabewerte.

Abbildung 6.8 zeigt die konkurrierende Ausführung zweier MLT, die jeweils einen bestimmten Geldbetrag von einem Konto a auf ein Konto b transferieren. Mit Blick auf die r/w-Operationen auf der L_0-Ebene ist der angegebene Schedule nicht serialisierbar, jedoch können in diesem Fall die beiden *"deposit"*-Operationen auf der L_1-Ebene als kommutativ betrachtet werden, so daß der L_0-Konflikt auf b zu einem *Pseudokonflikt* wird und der Schedule auf der L_1-Schicht als (mehrschichtig) serialisierbar angesehen werden kann.

Bei der Begründung der "High-Level"-Serialisierbarkeit des Beispielschedules aus der Abbildung 6.8 wurde implizit angenommen, daß jede Operation auf einer höheren Abstraktionsebene als elementar, d.h. unteilbar, im Hinblick auf eine konkurrierende Abarbeitung angesehen wird. Da jedoch eine L_1-Operation aus mehreren L_0-Operationen

bestehen kann und die L_0-Operationen zweier L_1-Aktionen durchaus "verzahnt" ausgeführt werden können, ist diese Forderung nicht automatisch garantiert. Folglich muß allgemein gesichert werden, daß in jeder Ebene L_i ein Synchronisationsmechanismus arbeitet, der die L_i-Operationen als Aktionen von Subtransaktionen betrachtet, die jeweils eine L_{i+1}-Operation realisieren. Ziel eines L_i-Synchronisationsmechanismus ist es also, die Subtransaktionen der L_{i+1}-Aktionen voneinander zu isolieren. Präziser ausgedrückt, realisiert der Synchronisationsmechanismus der Schicht L_i die Serialisierbarkeit des L_i-Schedules, wobei die Subtransaktionen der L_{i+1}-Aktionen die Transaktionen des Schedules bilden und CON_i die Konflikte der jeweiligen Ebene enthält.

Formal läßt sich das MLT-Modell folgendermaßen zusammenfassen: Eine MLT in einem System mit n Ebenen $L_0, \ldots, L_{n-1}$ ist definiert als ein Transaktionsbaum der Höhe $n+1$ mit allen Blattknoten in L_0. Die Knoten des Baumes werden Aktionen genannt und repräsentieren die Ausführung ebenenspezifischer Operationen. Kanten zwischen Aktionen der Ebenen L_i und L_{i-1} repräsentieren die implementationsspezifische Expansion einer L_i-Aktion in L_{i-1}-Operationen. Ein *mehrschichtiger Schedule* (MLS – "multi level schedule") ist durch eine Menge von Transaktionsbäumen und eine Ausführungsreihenfolge $\prec_0$ der ausgeführten L_0-Operationen definiert. Die Reihenfolge $\prec_0$ enthält dabei zumindest alle L_0-Aktionspaare, die in CON_0 enthalten sind. Nimmt man das r/w-Modell als Basis in L_0[6] an, enthält $\prec_0$ zumindest alle Aktionen die miteinander in einem rw-, wr- oder ww-Konflikt stehen. Die Ausführungsreihenfolge $\prec_i$ einer höheren Ebene wird aus $\prec_0$ folgendermaßen bestimmt: Eine L_i-Aktion f geht genau dann einer L_i-Aktion g voran ($f \prec_i g$), wenn alle L_0-Aktionen von f denen von g vorangehen. Sind sie dagegen "verzahnt", dann sind f und g nicht in einer $\prec_i$-Beziehung.

Aus den Relationen $\prec_i$ und CON_i läßt sich die *L_i-Serialisierungsreihenfolge* $\overset{<}{\sim}_i$ wie folgt bestimmen: In L_0 enthält $\overset{<}{\sim}_0$ alle geordneten Konfliktpaare, d.h. $\overset{<}{\sim}_0 = \prec_0 \cap CON_0$. Für zwei L_i-Aktionen f und g gilt genau dann $f \overset{<}{\sim}_i g$, wenn es in L_{i-1} von f und g erzeugte Schritte f' und g' in CON_{i-1} gibt, für die $f' \overset{<}{\sim}_{i-1} g'$ gilt. Ein MLS ist genau dann *mehrschichtig serialisierbar* (MLSR – "multi level serializable"), wenn in jeder Ebene L_i ($0 \leq i \leq n$) gilt, daß die Relation $\overset{<}{\sim}_i \cup (\prec_i \cap CON_i)$ azyklisch ist. Die wichtige praktische Bedeutung der MLSR-Theorie äußert sich wie folgt [SWS91, WS92]: Angenommen, daß für jedes L_{i+1}-Operationspaar (f, g) aus CON_{i+1} mindestens ein Paar L_i-Operationen (f', g') unter den von f bzw. g erzeugten Schritten existiert, das in CON_i enthalten ist, dann kann mehrschichtige Serialisierbarkeit durch die Eigenschaft der *"Level-by-level"-Serialisierbarkeit* gewährleistet werden. Diese fordert, daß jeweils der Serialisierbarkeitsgraph zwischen zwei benachbarten Ebenen azyklisch sein muß. Praktisch bedeutet dies, daß unterschiedliche Synchronisationsmechanismen auf verschiedenen Ebenen genutzt werden können, solange sie die Serialisierbarkeit des Schedules der entsprechenden Ebene gewährleisten.

MLT-Management gewährleistet eine höhere Parallelität der Transaktionen im Vergleich zu klassischen (einschichtigen) Synchronisationsmechanismen, da Ergebnisse in

[6] Nicht alle der angegebenen Publikationen gehen von diesem Modell als Basis aus, beispielsweise verwenden [DSW94] "update"-Operationen als Elemente von L_0.

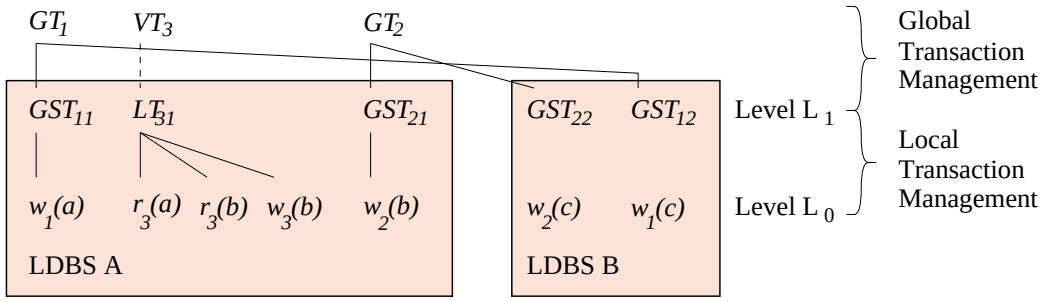

Abbildung 6.9: Beispiel eines nichtserialisierbaren FDBS-Schedules

tieferen Schichten früher freigegeben und damit für parallel laufende Transaktionen sichtbar werden. Dies führt jedoch auch dazu, daß Recovery-Maßnahmen beim Abbruch einer MLT nicht durch einfaches Rücksetzen auf den Zustand vor dem Beginn der Transaktionsausführung realisiert werden können. Eine Lösung dieses Problems bietet das Kompensieren von kompletten Subtransaktionen mittels inverser "High-Level"-Operationen. Betrachtet man die Kompensationstransaktionen als regulären Teil der abzubrechenden MLT, können auch Transaktionsabbrüche im Kontext der mehrschichtigen Serialisierbarkeit behandelt werden. Das resultierende Kriterium ist die *komplette Serialisierbarkeit*, die die MLSR für Schedules gewährleistet, in denen kompensierende Subtransaktionen explizit präsent sind. Die komplette Serialisierbarkeit bietet somit als einfache Erweiterung der MLSR die Möglichkeit, Synchronisations- und Recovery-Aspekte mit einem einzigen Korrektheitskriterium zu behandeln. Die allgemeine Darstellung des MLT-Modells soll an dieser Stelle beendet werden, ausführlichere Darstellungen dieses Ansatzes finden sich in [WS92, VG93].

In [SWS91] wird ein allgemeiner Versuch unternommen, ein föderiertes Transaktionsmanagement zu entwickeln, das auf MTL basiert. Besonders Augenmerk wird dabei auf lokale Transaktionen gerichtet, die im MLT-Modell nicht zu finden sind und das System quasi durch eine "Seitentür" betreten und erst auf tieferen Abstraktionsebenen im MLT-Modell auftauchen. Das Problem ist dabei das Feststellen derjenigen Informationen über die lokalen Transaktionen, die es dem GTM möglich machen, mit den Interaktionen von globalen Subtransaktionen und lokalen Transaktionen zurechtzukommen. Das MLT-Modell wird dabei wie folgt als Transaktionsmodell eines FDBMSs verwendet: In jedem LDBMS bilden die lokalen Transaktionen und globalen Subtransaktionen die höhere Ebene L_1, L_0 wird durch die entsprechenden Zugriffsoperationen gebildet. Im Gegensatz zum MLT-Modell hat L_1 dabei nicht unbedingt eine höhere Semantik als L_0. Es wird jedoch davon ausgegangen, daß in L_1 die Serialisierungsreihenfolge innerhalb des CDBSs bekannt ist und dem GTM zur Entscheidung, ob der Schedule global serialisierbar ist, propagiert werden kann. Dabei wird die dynamische Konfliktrelation $DCON_1$ als transitive Hülle der Serialisierungsreihenfolge $\overset{<}{\sim}_1$ gebildet. Mit Hilfe dieser dynamischen Konfliktrelation ist eine Betrachtung der MLSR in einer föderierten Umgebung möglich.

Abbildung 6.9 zeigt ein in diesem Sinn nichtserialisierbaren Schedule. Die "virtuelle Transaktion" (VT) agiert dabei als formaler Repräsentant der lokalen Transaktion im globalen Schedule. Da in LDBS A die Abhängigkeit $GST_{11} \lesssim_1 LT_{31} \lesssim_1 GST_{21}$ gilt, ist $(GST_{11}, GST_{21}) \in DCON_1$ und folglich gilt $GT_1 \lesssim_2 GT_2$. In LDBS B gilt jedoch $GST_{22} \lesssim_1 GST_{12}$, d.h. $(GST_{22}, GST_{12}) \in DCON_1$ und damit $GT_2 \lesssim_2 GT_1$. Folglich ist der dargestellte Schedule mit einer derartig durchgeführten Definition von $DCON_1$ nicht (mehrschichtig) serialisierbar. In [SWS91] werden nun verschiedene Möglichkeiten untersucht, wie man die Definition der Konfliktoperation modifizieren kann, um semantische oder den Ausführungskontext betreffende Aspekte zu berücksichtigen. Bei der Definition von $DCON_1$ ist bislang davon ausgegangen worden, daß kein Wissen über die Kommutativität von L_1-Operationen vorliegt und alle Konflikte in $DCON_1$ ausschließlich auf deren lokaler Ausführung beruhen. Das heißt, $DCON_1$ enthält alle L_1-Aktionen, die in L_0 zueinander in Konflikt stehen. Jedoch können einige Konflikte zwischen globalen Subtransaktionen Pseudokonflikte sein.

Die erste durchgeführte Modifikation von $DCON_1$ betrifft den Ausführungskontext und untersucht, ob man in der Ausführungsreihenfolge $\prec_0$ die Aktionen zweier, nicht miteinander in einem *direkten* Konflikt stehender, globaler Subtransaktionen vertauschen kann, ohne die Serialisierungsreihenfolge zu ändern. Ist ein vollständiger Tausch der globalen Subtransaktionen möglich, so ist ein Konflikt zwischen diesen ein Pseudokonflikt. Man betrachte beispielsweise einen lokalen Schedule $r_{LT_{31}}(a) \prec_0 w_{GST_{11}}(a) \prec_0 w_{GST_{21}}(b) \prec_0 w_{LT_{31}}(b)$. In diesem Fall ist (GST_{11}, GST_{21}) aufgrund der Transitivität in $DCON_1$, aber man kann die Ausführungsreihenfolge der globalen Transaktionen vertauschen, ohne die Serialisierungsreihenfolge zu verändern. Bereinigt man $DCON_1$ um derartige Konflikte, erhält man die Kontextkonfliktrelation $CCON_1$ und mit ihr das Kriterium *"MLSR by context"*. Ohne die explizite Ausnutzung von Semantik erhält man so ein Kriterium, das der Quasiserialisierbarkeit entspricht und das schwächste Kriterium ist, das man ausschließlich durch Ausnutzung von L_0-Kommutativität realisieren kann. Wenn man dagegen semantische Aspekte nutzt, d.h. wenn man die Kommutativität von Aktionen auf der Ebene L_1 definiert, dann lassen sich noch schwächere Kriterien entwickeln, die entsprechend mehr Schedules zulassen. Definiert man beispielsweise die beiden Subtransaktionen GST_{12} und GST_{22} in der Abbildung 6.9 als "deposit"-Operationen, die kommutativ sein sollen, so erfüllt der dargestellte Schedule die Anforderungen an die *semantische MLSR*. Die Definition der entsprechenden Konfliktrelation für dieses und weitere Kriterien sind in [SWS91] dargestellt.

[SWS91] betrachten weiterhin das Kriterium der kompletten Serialisierbarkeit in föderierten Umgebungen. Dabei wird davon ausgegangen, daß bei Abbrüchen von lokalen Transaktionen ein zustandsbasiertes Rücksetzen der einzige anzuwendende Recovery-Mechanismus ist, während für den Abbruch globaler Transaktionen angenommen wird, daß genügend semantisches Wissen für eine Kompensation vorhanden ist. In diesem Fall liefert die komplette Serialisierbarkeit ein geeignetes Kriterium, um fehlerhafte Interaktionen von Synchronisations- und Recovery-Mechanismen in föderierten Umgebungen zu vermeiden.

In [SWS91] wird abschließend ein Protokoll vorgestellt, mit dem semantische MLSR in FDBSen sichergestellt werden kann. Dabei wird ein geschachteltes 2PL-Protokoll ("nested two phase locking") verwendet, das das herkömmliche 2PL-Protokoll um semantische Sperren erweitert und in den verschiedenen Ebenen angewendet werden kann. Eine globale Subtransaktion fordert dabei sowohl lokale Sperren in der Ebene L_0 als auch semantische Sperren in L_1 an. Die lokalen Sperren werden mit dem Ende der globalen Subtransaktion freigegeben, die sematischen Sperren werden dagegen bis zum Ende der globalen Transaktion gehalten. Lokale Transaktionen werden in diesem Protokoll wie folgt behandelt: Die Grundidee besteht darin, lokale Transaktionen, die transitive Beziehungen zwischen zwei globalen Subtransaktionen erzeugen könnten, in ihrer Ausführung zu verzögern. Da die Erkennung von Konflikten, in die lokale Transaktionen involviert sind, nur in der Schicht L_0 möglich ist, ist es nötig, eine Möglichkeit zu schaffen, die die L_0-Sperren globaler Subtransaktionen nach deren Abarbeitungsende bis zum Ende der globalen Transaktion zurückhält ("retained L_0-locks"). Zur Implementation dieser Idee wird davon ausgegangen, daß alle CDBMSe in der Lage sind, lokale Transaktionen von globalen Subtransaktionen zu unterscheiden. Mit dem Ende einer globalen Subtransaktion werden dann alle von ihr gehaltenen Sperren zu "retained locks". Lokalen Transaktionen ist es dann nicht möglich, eine Sperre für ein Datenobjekt zu erhalten, für das noch eine solche zurückgehaltene Sperre existiert. Sperranforderungen anderer globaler Subtransaktionen erzeugen dagegen keine Konflikte mit den zurückgehaltenen Sperren, d.h. globalen Subtransaktionen ist es möglich, die zurückgehaltene Sperre eines Datenobjektes selbst zu erwerben. Mit dem Ende einer globalen Transaktion werden alle noch gehaltenen "retained locks" freigegeben.

In [SS93, WDSS93] wird von dem etwas allgemeineren ONT-Ansatz ausgegangen, jedoch sind die dargestellten Modelle durchaus mit den bisher gemachten Ausführungen vereinbar. Beide Artikel gehen davon aus, daß alle an einer Föderation partizipierenden CDBSe eine bestimmte Menge von "High-Level"-Operationen, z.B. SQL-Operationen, exportieren, die in der Ebene L_1 zum Aufruf durch lokale Transaktionen und globale Subtransaktionen zur Verfügung stehen. Der Aufruf der exportierten L_1-Operationen wird in die ONT-Strukturen des föderierten Systems eingebettet. Die Administratoren der einzelnen Datenbankknoten sind für die Definition der Konfliktrelation der jeweiligen L_1-Schicht verantwortlich. In [SS93, WDSS93] findet ebenfalls das oben dargestellte Sperrprotokoll Verwendung.

[SS93] stellen dabei eine Systemarchitektur vor, die zur Umsetzung des Sperrprotokolls für globale Subtransaktionen und lokale Transaktionen in der L_1-Schicht geeignet erscheint und die in Abbildung 6.10 dargestellt wird. In dieser Architektur werden für alle CDBSe einer Föderation *"Agenten"* implementiert, die ein zum zugehörigen Datenbanksystem identisches Interface besitzen. Die Aufgabe eines Agenten besteht darin, die lokalen Transaktionen und globalen Subtransaktionen zu kontrollieren und ihre Ausführung zu koordinieren. Folglich sind auch alle lokalen Transaktionen gezwungen, die Datenbanksysteme über die Agenten zu "betreten". Die Agenten sind für das Transaktionsmanagement der L_1-Ebene verantwortlich, d.h. die L_1-Operationen der globalen Subtransaktionen und lokalen Transaktionen werden auf Konflikte überprüft und ihre

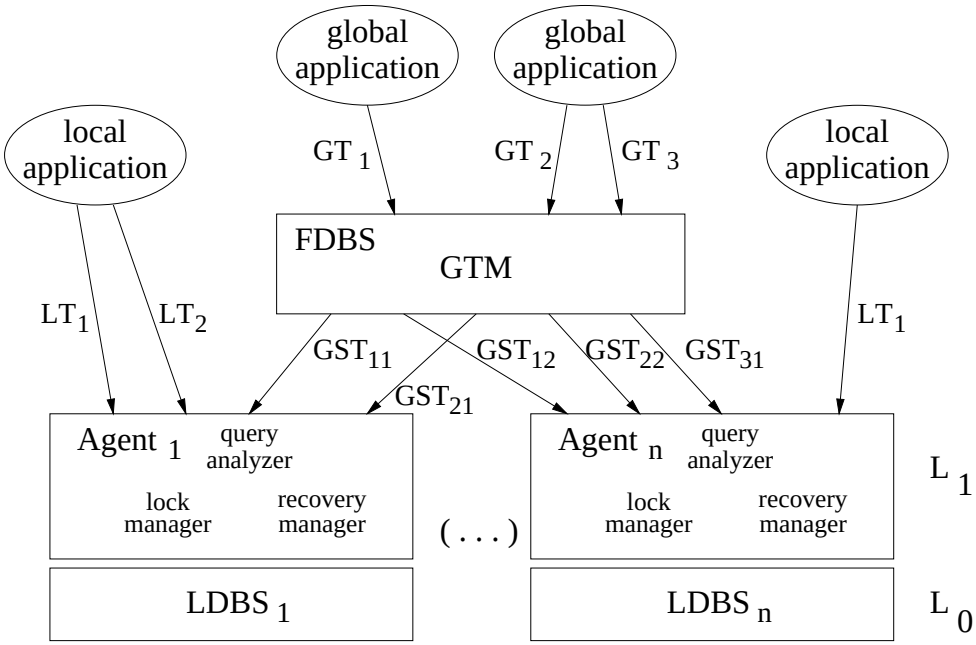

Abbildung 6.10: Architektur eines FDBS-Prototyps

Abarbeitung entsprechend gesteuert. Die Konflikterkennung kann durch den Agenten mittels der definierten Konfliktrelation und geeigneter Prädikatvergleiche durchgeführt werden, zur Steuerung der Abarbeitung wird obiges Sperrprotokoll forciert. Eine detaillierte Erläuterung der Arbeitsweise und des Aufbaus eines solchen Agenten und ihr Zusammenwirken mit den globalen Komponenten der Föderation wird in [SS93] vorgenommen. [SS93, WDSS93] weisen darauf hin, daß der ONT-Ansatz beim föderierten Transaktionsmanagement besonders in lose gekoppelten Umgebungen zur Anwendung kommen kann.

Mit der Anwendung eines semantikbasierten MLT-Managements in föderierten Datenbanksystemen beschäftigt sich auch [DSW94]. In dieser Publikation wird ebenfalls davon ausgegangen, daß die einzelnen CDBSe eine bestimmte Menge von "High-Level"-Operationen exportieren, die den globalen Transaktionen als *exportierte Subtransaktionen* zum Aufruf zur Verfügung stehen. [DSW94] betrachten besonders den Einfluß von semantischen Aspekten auf die Synchronisations- und Recoverymechanismen für MLT in FDBSen. Für die Synchronisation von MLT wird dabei eine neue Formulierung für die Korrektheit eines MLS gegeben, die die Kompatibilität von Operationen auf unterschiedlichen Abstraktionsebenen ausnutzt. Ein mehrschichtiger Schedules ist demnach dann *semantisch serialisierbar*, wenn er durch die Anwendung der folgenden Regeln schrittweise in einen seriellen Schedule überführt werden kann [DSW94, WDSS93]:

1. *Kompatibilität von Operationen*: Zwei $\prec_i$-geordnete benachbarte Unterbäume können ihre $\prec_i$-Reihenfolge vertauschen, wenn die Wurzeln der Unterbäume kompatible Operationen sind, die zu verschiedenen Transaktionsbäumen gehören.

2. *Reduktion von isolierten Unterbäumen*: Ein isolierter Unterbaum kann auf seine Wurzel reduziert werden, wobei ein Unterbaum isoliert ist, wenn alle seine Knoten in tieferen Ebenen seriell sind und der gesamte Unterbaum $\prec_i$-geordnet ist, d.h. der Unterbaum ist nicht mit anderen Unterbäumen "verzahnt".

Mittels dieser beiden Regeln läßt sich die semantische MLSR eines Schedules nachweisen. Betrachtet man zusätzlich die Recovery-Maßnahmen in einem MLS, d.h. das zustandsbasierte Rücksetzen von lokalen Transaktionen und Kompensieren abgeschlossener Subtransaktionen zur Realisierung globaler Transaktionsabbrüche, dann muß eine dritte Regel angewendet werden, um die *komplette Serialisierbarkeit* zu erhalten:

3. *Kompensation von Operationen*: Wenn die Kompensationstransaktion f^{-1} der ursprünglichen Operation f einer globalen Subtransaktion sofort auf f im Schedule folgt und f und f^{-1} sind isoliert, dann können f und f^{-1} aus dem Transaktionsbaum entfernt werden.

Kann ein mehrschichtiger Schedule unter Anwendung dieser drei Regel schrittweise in einen seriellen Schedule überführt werden, so ist er komplett serialisierbar. In [DSW94] wird weiterhin ein Prototyp eines FDBMSs vorgestellt, der innerhalb des COSMOS-Projektes an der ETH Zürich entwickelt wurde. Der Scheduler der MLT arbeitet dabei zweistufig: Zunächst werden auf der Ebene L_1 die globalen Subtransaktionen in Form der exportierten Subtransaktionen durch einen semantischen Scheduler synchronisiert. Die lokalen Transaktionen erscheinen erst auf der Ebene L_0. [DSW94] gehen, wie bereits erwähnt, von höheren Operationen, wie z.B. SQL-Operationen, auf der Ebene L_0 aus. Auf dieser Ebene arbeitet ein DML-Scheduler, der die Operationen in der Anfragesprache (DML – "data manipulation language") des jeweiligen CDBMSs mittels des oben aufgeführten 2PL-Protokolls synchronisiert.

Der in [DSW94] vorgeschlagene Korrektheitsansatz ist in einem engen Zusammenhang mit dem in [SWY93, AVA+94] vorgestellten Korrektheitskriterium der *Reduzierbarkeit* (RED – *"reducibility"*) zu sehen. Beide Publikationen versuchen, ein einheitliches Kriterium für Synchronisations- und Recovery-Mechanismen zu definieren, das weniger restriktiv als die Rigorosität von Schedules ist, aber dennoch die Atomarität und Serialisierbarkeit globaler Transaktionen sichert. Die dabei aufgestellten und untersuchten Kriterien gelten nicht nur für MLT/ONT-Ansätze, sondern genauso für "flache" Transaktionen. Die Kommutativität bestimmter Transaktionsoperationen und die Möglichkeit der Realisierung von Recovery-Mechanismen mittels inverser Operationen bilden die Grundlage der angestellten Betrachtungen. Beide Artikel untersuchen erweiterte Schedules ("expanded Schedules"), d.h. Schedules, in denen eventuell notwendige Recovery-Aktionen explizit präsent sind. Ein erweiterter Schedule heißt in [SWY93, AVA+94] reduzierbar, wenn er mit Hilfe dreier Regel, die inhaltlich den oben genannten entsprechen, in einen seriellen Schedule überführbar ist. Gilt diese Eigenschaft auch für alle Präfixe des Schedules, so ist dieser *präfix-reduzierbar* (PRED – *"prefix-reducible"*). In [SWY93, AVA+94] wird argumentiert, daß die Klasse der PRED-Schedules die am wenigsten restriktive Klasse von Ausführungen ist, die intuitiv korrekt im Hinblick auf

die Synchronisations- und Recovery-Mechanismen ist. Details zur Definition dieses und weiterer Kriterien sind [SWY93, AVA+94] zu entnehmen.

In [SSW95] wird ebenfalls eine Implementierung eines Prototyps für ein föderiertes Transaktionsmanagement, das auf mehrschichtigen Transaktionen beruht, vorgestellt (vergleiche Abbildung 6.10) und Performance-Untersuchungen am System durchgeführt. Dabei realisiert ein GTM die Serialisierbarkeit der globalen Transaktionen mit Blick auf eine gegebene globale Konfliktrelation. Die Synchronisation der globalen Subtransaktionen wird durch Agenten, ähnlich denen der vorangegangenen Ansätze, realisiert. Dazu werden lokale Konfliktrelationen definiert, wobei davon ausgegangen wird, daß für jeden Konflikt in der globalen Konfliktrelation zumindest ein Konflikt in der lokalen Konfliktrelation existiert. Damit kann die globale Serialisierbarkeit durch die Serialisierbarkeit des Schedules auf der globalen und der lokalen Ebene realisiert werden ("Level-by-Level"-Serialisierbarkeit), ohne weitere Restriktionen an die Art der Scheduler vorzunehmen. Bei der Implementierung des Prototyps wurden einige vereinfachende Annahmen zugrundegelegt. Der Prototyp geht von SQL-Interfaces der lokalen Datenbanksysteme aus, dementsprechend arbeiten auch die Agenten mit dieser DML. Jedem globalen Datenobjekt ist genau ein lokales Datenobjekt zugeordnet und umgekehrt. Gleiches gilt für die globalen und lokalen Operationen. Folglich müssen keine Transformationen der globalen Operationen vorgenommen werden. Außerdem wird davon ausgegangen, daß ausschließlich globale Transaktionen im System arbeiten. Unter diesen Annahmen wurden auf einer Testdatenbank Performance-Untersuchungen durchgeführt, die den MLT-Ansatz mit dem klassischen 2PL/2PC-Ansatz vergleichen. Bei allen durchgeführten Messungen ergaben sich Leistungssteigerungen beim Einsatz von MLT-Techniken gegenüber dem klassischen Ansatz.

Die beiden letzten Papiere, die hier erwähnt werden sollen, sind [MR91, MVN93], die im Zuge der Entwicklung des objektorientierten Datenbanksystems VODAK durch das GMD IPSI veröffentlicht wurden. Beide Artikel beschäftigen sich ebenfalls mit dem Einsatz mehrschichtiger Transaktionen in verteilten, heterogenen und autonomen Datenbanksystemen. In [MR91] werden verschiedene Möglichkeiten einer atomaren Terminierung von globalen Transaktionen in derartigen Umgebungen untersucht und man kommt zu dem Ergebnis, daß die lokale Terminierung von globalen Subtransaktionen *vor* einer globalen Entscheidungsfindung, wie sie in Abbildung 6.7 dargestellt wurde, in einem föderierten Transaktionsmanagement am besten mit dem MLT-Ansatz vereinbar ist. In [MVN93] wird ein erweitertes MLT-Modell vorgestellt, das speziell die Heterogenität und Autonomie der in einer Föderation beteiligten Systeme unterstützen soll. Dazu werden in das MLT-Modell Aspekte wie die verteilte Ausführung globaler Transaktionen und autonome lokale Transaktionen eingeführt. Die Schnittstelle zwischen den CDBSen und dem globalen System wird dabei durch eine Menge durch das CDBMS *exportierter lokaler Transaktionsprogramme* modelliert. Durch die Definition bestimmter Untermengen dieser exportierten lokalen Transaktionsprogramme, die von globalen Transaktionen als Subtransaktionen ausgeführt werden dürfen, und die Anwendung semantikbasierter Konfliktrelationen gelingt es, das Modell für unterschiedliche Anforderungen an die lokale Autonomie und globale Konsistenzerhaltung zu parametrisieren. Die Details der neuge-

faßten Begriffsbestimmungen und Lösungsansätze zur Konfliktbehandlung sind [MVN93] zu entnehmen.

Alle hier vorgestellten Ansätze haben eines gemeinsam: Sie gehen davon aus, daß man die Anforderungen an eine föderierte Transaktionsverwaltung gut mit den, durch den Einsatz von MLT/ONT-Modellen zur Verfügung gestellten Eigenschaften realisieren kann. Sowohl bei der Betrachtung der Synchronisationsmechanismen in einem föderierten System, als auch bei der Durchsetzung von Recovery-Maßnahmen können durch die Ausnutzung von Semantik in höheren Abstraktionsebenen Probleme auf der unteren physischen Ebene, wie das Serialisierbarkeit- oder Atomaritätsproblem aus Kapitel 3, gelöst werden. Gleichzeitig kann der Einsatz von MLT/ONT-Ansätzen in einer Föderation durch vorzeitiges Freigeben von Sperren verbunden mit der Anwendung von Kompensationstechniken zu einem höheren Transaktionsdurchsatz und einer gesteigerten Parallelität der Abarbeitung führen.

An dieser Stelle soll die Betrachtung der Lösungsmöglichkeiten zur Sicherung der Atomarität und Persistenz globaler Transaktionen im allgemeinen und die Anwendung von Kompensationstechniken zu deren Realisierung im speziellen beendet werden, um nachfolgend das dritte, in föderierten Umgebungen zu lösende Problem zu betrachten: Die Sicherung verklemmungsfreier globaler Abarbeitungen.

Kapitel 7

Globale Verklemmungen

In den vorangegangenen Kapiteln wurde untersucht, wie das globale Serialisierbarkeitsproblem und das globale Atomaritäts- und Fehlertoleranzproblem in föderierten Umgebungen gelöst werden können. Innerhalb dieses Kapitels sollen nun Ansätze und Methoden vorgestellt werden, die bei der Lösung des globalen Verklemmungsproblems angewandt werden können. Wie bereits im Abschnitt 3.3.3 einleitend erläutert wurde, versteht man unter einer *Verklemmung* ("deadlock") eine Situation, in der ein Menge von Transaktionen sich gegenseitig derart blockiert, daß ohne den Abbruch einer oder mehrerer Transaktionen aus dieser Menge keine der Transaktionen zu einem erfolgreichen Abarbeitungsende gelangen kann. Solche Verklemmungssituationen treten häufig in zentralen Datenbanksystemen auf, die sperrende Synchronisationsverfahren, wie z.B. 2PL-Verfahren, benutzen. Im Abschnitt 3.3.3 wurde bereits an einem Beispiel gezeigt, daß in föderierten Umgebungen auch dann globale Verklemmungssituationen auftreten können, wenn alle lokalen Abarbeitungen verklemmungsfrei sind. Das Problem liegt dabei darin, daß in bestimmten Fällen für die Fortführung oder Beendigung einer globalen Transaktion in einem Datenbankknoten des FDBSs die Ausführung einer Operation in einem anderen Knoten abgewartet werden muß. Es besteht also die Möglichkeit "globaler Wartezustände" über die Knotengrenzen einzelnen CDBSe hinweg.

Bei der Analyse von Verklemmungssituationen werden in der Regel *"wait-for"-Graphen* (WFG) eingesetzt, die die jeweiligen Transaktionen als Knotenmenge verwalten und genau dann eine Kante zwischen einer Transaktion T_i und T_j enthalten ($T_i \rightarrow T_j$), wenn zur weiteren Ausführung die Transaktion T_i auf bestimmte Ereignisse, z.B. das Freigeben einer Sperre, in der Ausführung von T_j warten muß. Eine Verklemmungssituation tritt in einem System genau dann ein, wenn der WFG des Systems einen Zyklus enthält. Für das Beispiel 3.4 sehen die Graphen der einzelnen Systeme folgendermaßen aus: Im Knoten s_1 wartet LT_3 auf eine Sperrenfreigabe der Transaktion GT_1, GT_2 wartet dagegen auf eine Sperre, die von LT_3 gehalten wird. Der lokale WFG des Knotens s_1 enthält damit die Abhängigkeiten: $GT_2 \rightarrow LT_3 \rightarrow GT_1$. Im Knoten s_2 wartet GT_1 auf die lokale Transaktion LT_4, die ihrerseits auf GT_2 wartet. Der lokale WFG dieses Knotens enthält dementsprechend folgende Kanten: $GT_1 \rightarrow LT_4 \rightarrow GT_2$. Beide lokalen WFG sind offensichtlich zyklenfrei, so daß lokal verklemmungsfreie Abarbeitungen vor-

liegen. Es bestehen jedoch Abhängigkeiten über die Grenzen der lokalen Knoten hinweg. So kann GT_1 in s_1 erst dann die Sperren freigeben, wenn die globale Transaktion auch in s_2 erfolgreich abgearbeitet wurde. Gleiches läßt sich für GT_2 feststellen. Ein globaler WFG kann gebildet werden, indem die lokalen WFG vereinigt werden. Für Beispiel 3.4 sieht der globale WFG wie folgt aus: $GT_1 \rightarrow LT_4 \rightarrow GT_2 \rightarrow LT_3 \rightarrow GT_1$. Dieser Graph enthält offensichtlich einen Zyklus, ein globaler Deadlock liegt in diesem Beispiel vor.

In zentralen wie in verteilten Systemen sind drei grundsätzliche Herangehensweisen bei der Behandlung von Verklemmungssituationen bekannt: die Verklemmungsprävention, die Verklemmungsvermeidung und die Verklemmungserkennung und -auflösung. In [ÖV91] werden speziell verteilte Mechanismen für alle drei Lösungsansätze dargestellt, deren Eignung für föderierte Systeme nun genauer untersucht werden soll.

7.1 Mechanismen zur Verklemmungsprävention

Mechanismen zur Verklemmungsprävention sichern schon zum Ausführungsbeginn einer Transaktion, daß keine Verklemmungen während der Ausführung auftreten. Der Transaktionsmanager überprüft schon zu Beginn der Ausführung, ob eine Transaktion zu Verklemmungen führen kann und verhindert die Ausführung, wenn dies der Fall ist. Um so eine Überprüfung durchzuführen, ist es erforderlich, daß alle Datenobjekte, auf die zugegriffen werden soll, schon zum Beginn der Ausführung vordeklariert sind. Folglich ist es nicht möglich, in Abhängigkeit von bestimmten gelesenen Datenobjekten im Ausführungsverlauf zu entscheiden, auf welche Datenobjekte als nächstes zugegriffen werden soll. Es muß also bei Transaktionen ohne eine feste lineare Zugriffsstruktur die maximale Menge der eventuell referenzierten Datenobjekte zur Verklemmungsprävention vordeklariert werden. Sind zum Ausführungsbeginn alle Datenobjekte verfügbar, reserviert der Transaktionsmanager diese Objekte für die Transaktion und gestattet deren Fortführung, ansonsten wird eine weitere Ausführung der Transaktion untersagt.

Die Anwendung von Präventionsmechanismen scheint allgemein ziemlich unpraktikabel. Demnach ist sie auch in föderierten Umgebungen kaum einsetzbar. Selbst wenn alle lokalen Systeme mit Präventionsmechanismen ausgestattet sind, sind globale Verklemmungen möglich. Der GTM muß also selbst Mechanismen zur globalen Verklemmungsprävention anwenden. Dies ist jedoch nicht ohne die Kenntnis der Vordeklarationen sämtlicher lokaler Transaktionen möglich, was wiederum der Forderung nach lokaler Autonomie widerspricht. Außerdem schränkt die Vordeklaration von Datenobjekten die Struktur der verarbeitbaren Transaktionen ein und die Angabe von Maximalmengen führt zu einer Verringerung der Parallelität. Andererseits verlangen derartige Mechanismen keine Unterstützung während der Transaktionslaufzeit und Abbrüche und Neustarts von Transaktionen durch Verklemmungen werden vermieden.

7.2 Mechanismen zur Verklemmungsvermeidung

Verklemmungsvermeidende Ansätze verwenden dagegen Synchronisationsmechanismen, die keine Verklemmungen erzeugen können oder diese im vorraus erkennen und vermeiden. Der einfachste Ansatz dabei ist dabei, eine totale Ordnung der Datenressourcen aufzustellen und zu fordern, daß Transaktionen ihre Zugriffsanforderungen in der entsprechenden Reihenfolge stellen. Für verteilte Systeme bedeutet dies, daß entweder eine globalen oder diverse lokale Reihenfolgen aufgestellt werden, im letzteren Fall muß auch eine Ordnung der teilnehmenden Knoten erfolgen.

Andere Ansätze zur Verklemmungsvermeidung gehen von Zeitstempeln zur Priorisierung von Transaktionen aus und vermeiden Verklemmungen, indem sie Transaktionen mit höheren oder niedrigeren Prioritäten abbrechen. Die bekanntesten Algorithmen basieren auf der *"wait-die"*- oder der *"wound-wait"*-Regel, die folgendermaßen lauten:

- **"wait-die"-Regel** : Fordert eine Transaktion T_i die Sperre eines Datenobjektes an, das durch eine Transaktion T_j gesperrt ist, dann ist es T_i erlaubt zu warten, wenn T_i älter als T_j ist, d.h. $ts(T_i) < ts(T_j)$ gilt. Ist T_i dagegen jünger, wird T_i abgebrochen und mit demselben Zeitstempel erneut gestartet.

- **"wound-wait"-Regel** : Fordert eine Transaktion T_i die Sperre eines Datenobjektes an, das durch eine Transaktion T_j gesperrt ist, dann ist es T_i erlaubt zu warten, wenn T_i jünger als T_j ist, d.h. $ts(T_i) > ts(T_j)$ gilt. Ist T_i dagegen älter, wird T_j abgebrochen ("wounded") und T_i erhält die Sperren sofort.

Beide Algorithmen brechen in Verklemmungssituationen die jeweils jüngeren Transaktionen ab. Dennoch bevorzugt der "wait-die"-Ansatz jüngere Transaktionen, da solange keine Verklemmungen auftreten, ältere Transaktionen auf jüngere warten müssen. Der "wound-wait"-Ansatz dagegen bevorzugt ältere Transaktionen, da jüngere Transaktionen sofort abgebrochen werden, wenn eine ältere Transaktion eine konkurrierende Sperre anfordert.

Für ein föderiertes System muß bei der Anwendung dieser Ansätze gefordert werden, daß die Zeitstempel lokal eindeutig sind, die Zeitstempel globaler Transaktionen zentral vergeben werden, d.h. alle Subtransaktionen einer globalen Transaktion haben den gleichen Zeitstempel und alle lokalen Systeme realisieren entweder den "wait-die"- oder den "wound-wait"-Algorithmus. Bei der Kombination der Ansätze sind weiterhin Verklemmungen möglich, wie man sich an Beispiel 3.4 klarmachen kann. Man nehme beispielsweise an, daß im Knoten s_1 ein "wound-wait"-Algorithmus verwendet wird, während s_2 einen "wait-die"-Ansatz realisiert. Nimmt man weiterhin an, daß der globalen Transaktion GT_1 der globale Zeitstempel $ts(GT_1) = 1$ vergeben wurde, daß beide lokalen Transaktionen die Zeitstempel $ts(LT_3) = 2$ und $ts(LT_4) = 2$ haben und der Zeitstempel der globalen Transaktion GT_2 in beiden Knoten $ts(GT_2) = 3$ ist, so kann in s_1 weiterhin GT_2 auf LT_3 und LT_3 auf GT_1 warten, während in s_2 die Transaktion GT_1 auf LT_4 und LT_4 auf GT_2 wartet. Ein globaler Deadlock ist nach wie vor möglich.

Genauso läßt es sich aber auch klarmachen, daß die Verwendung der gleichen Ansätze in allen Systemen die Verklemmungssituation auflöst. Die Ansätze sind demnach geeignet, um in verteilten Systemen während der Laufzeit eine effektive Verklemmungsvermeidung durchzuführen. Im Hinblick auf föderierte Umgebungen haben sie jedoch entscheidende Nachteile. Zum einen muß gefordert werden, daß alle teilnehmenden Datenbanksysteme den gleichen Verklemmungsvermeidungsalgorithmus verwenden, andererseits müssen die lokalen Systeme global vergebene Zeitstempel übernehmen. Beide Forderungen verletzen die lokale Autonomie der CDBMSe.

7.3 Mechanismen zur Verklemmungserkennung und -auflösung

Die Erkennung und Auflösung von Verklemmungssituationen ist der wohl populärste und meist-untersuchteste Ansatz zur Lösung des Verklemmungsproblems. Die Erkennung globaler Verklemmungen wird durch die Untersuchung des globalen WFG erreicht, indem man in dem globalen WFG Zyklen und die daran beteiligten Transaktionen sucht. Die Auflösung der Verklemmungen geschieht dann dadurch, daß eine oder mehrere Transaktionen ausgesucht werden, die als "Opfer" abgebrochen werden, um die entsprechenden Zyklen aufzubrechen. Die Auswahl derjenigen Transaktionen, deren Abbruch die geringsten Kosten verursacht, hat sich als schwer zu lösendes Problem (NP-vollständig) erwiesen, jedoch geben [BHG87] einige Anhaltspunkte, die eine Auswahl erleichtern:

1. Der Aufwand, der bereits in die Verarbeitung einer Transaktion investiert wurde. Dieser Aufwand geht bei einem Abbruch verloren.

2. Die Kosten, die ein Transaktionabbruch verursacht. Die Kosten hängen von der Anzahl der "update"-Operationen ab, die eine Transaktion bereits ausgeführt hat.

3. Der Aufwand, der nötig ist, um eine Transaktion zu beenden. Der Abbruch von Transaktionen, die kurz vor dem Bearbeitungsende stehen, sollte vermieden werden. Dies setzt jedoch voraus, daß der Scheduler das zukünftige Verhalten der aktiven Transaktionen vorhersagen kann.

4. Der Anzahl der Zyklen, in die eine Transaktion involviert ist. Da der Abbruch einer Transaktion alle Zyklen aufbricht, in denen die Transaktion erscheint, ist es das Beste, diejenigen Transaktionen zunächst abzubrechen, die in den meisten Zyklen erscheinen.

Nachdem hier einige Faktoren zur Verklemmungsauflösung genannt wurden, soll nun der Blick auf Mechanismen zur Verklemmungserkennung gerichtet werden. In [ÖV91] werden drei Ansätze zur Verklemmungserkennung beschrieben, die sich jeweils durch ihre Positionierung in einem verteilten System unterscheiden: Die zentrale, die hierarchische und die verteilte Verklemmungserkennung. Alle drei Ansätze gehen jedoch davon aus, daß

die teilnehmenden Systeme interne Synchronisations- und Blockierungsinformationen an über- oder gleichgeordnete Instanzen weitergeben. Konkret stehen die lokalen "wait-for"-Graphen an den Interfaces der einzelnen Systeme zur Verfügung. Dieser Ansatz ist in einem föderierten System jedoch nicht akzeptabel, da durch die lokale Autonomie die Systeme diese Infomationen nicht zur Verfügung stellen müssen. Es gilt also Alternativen zu finden, die einen Zugriff auf lokale "wait-for"-Informationen nicht nötig machen.

In [BLS91] wird ein Verklemmungserkennungsansatz für föderierte Umgebungen dargestellt, der auf einem globalen Graphen basiert, der "schätzungsweise" der Vereinigung der lokalen WFG entspricht. Der Ansatz arbeitet folgendermaßen: Der GTM reicht die Operation einer globalen Transaktion GT_i den Server im Knoten s_k weiter und wartet auf die Bestätigung der Ausführung. Erhält er die Bestätigung nicht, so kann GT_i in s_k in einem Wartezustand sein. Hat eine andere globale Transaktion GT_j Operationen auf s_k ausgeführt und wurde noch nicht beendet, dann kann GT_i direkt oder indirekt auf GT_j warten. Der GTM nimmt diese Beziehung an und addiert die Kante $GT_i \rightarrow GT_j$ in seinen "geschätzten" WFG. Solange dieser Graph zyklenfrei ist, ist auch die Abarbeitung verklemmungsfrei, tritt dagegen eine Verklemmungssituation ein, so enthält auch der "geschätzte" WFG einen Zyklus. Jedoch nicht jeder Zyklus im Graphen repräsentiert unbedingt eine Verklemmung. Bildet der "geschätzte" WFG einen Zyklus, obwohl gar keine globale Verklemmungssituation vorliegt, so wird vom GTM eine "falsche" Verklemmungssituation erkannt, die unnötige Abbrüche nach sich zieht. Um die Wahrscheinlichkeit "falscher" Verklemmungen und unnötiger Abbrüche zu verringern, kann der Ansatz mit Timeout-Mechanismen kombiniert werden. Die Kante $GT_i \rightarrow GT_j$ wird dann erst zum "geschätzten" WFG addiert, wenn die Ausführung der Operation von GT_i innerhalb eines festgelegten Timeout-Intervalls nicht bestätigt wurde.

Ein solcher Ansatz zur Verklemmungserkennung und -auflösung wird auch in [BST90, BST92] im Zusammenhang mit den dortigen Recovery-Mechanismen benutzt. Er soll an dieser Stelle vorgestellt werden.

a) Der "Potential-Conflict"-Graphen-Ansatz

An dieser Stelle soll die Erläuterung des graphenbasierten Ansatz von [BST90, BST92] aus dem Kapitel 6 fortgesetzt werden. Es gelten alle in Abschnitt 6.2 getroffenen Anforderungen und Aussagen. Nun soll untersucht werden, wie in einer solchen Umgebung mögliche globale Verklemmungssituationen geeignet vermieden werden können. Bei den allgemeinen Anforderungen für den graphenbasierten Ansatz ist davon ausgegangen worden, daß alle beteiligten Systeme lokalen Verklemmungen vorbeugen oder zumindest Mechanismen zur Erkennung und Auflösung lokaler Deadlocks enthalten. Trotz dieser lokalen Verklemmungsfreiheit sind im föderierten System eine Reihe von unterschiedlichen Situationen denkbar, die zu globalen Verklemmungen führen. Beispiel 3.4 gab ein Beispiel für eine solche Verklemmungssituation, die auch unter den hier herrschenden Bedingungen auftreten kann. Die folgenden Situationen sind ebenfalls möglich:

Beispiel 7.1 Gegeben sei die Situation aus dem Beispiel 6.1. Durch die Anwendung des Algorithmus zur sicheren Einreichung der "commit"-Operation aus Abschnitt 6.2 wurde die Kante $GT_2 \rightarrow GT_1$ in den WFCG eingefügt, GT_2 ist also in einem Wartezustand. Da die globale Transaktion GT_2 aber noch nicht beendet ist, hält sie weiterhin die Sperren für die Datenobjekte b und d in den Knoten s_1 bzw. s_2. Auf der anderen Seite kann die Redo-Transaktion RT_5 nicht gestartet werden, da die lokale Sperre für a von der lokalen Transaktion LT_3 gehalten wird, die ihrerseits auf die Freigabe der Sperre von b durch GT_2 wartet. Das System befindet sich in einer globalen Verklemmungssituation, da im WFCG die globale Transaktion GT_2 auf GT_1 wartet, während im lokalen System GT_1 auf LT_3 wartet, die wiederum das Ende von GT_2 abwarten muß. □

Beispiel 7.2 Ein FDBS bestehe aus zwei Knoten: s_1 mit den globalen Datenobjekten a und b, und s_2 mit dem globalen Datenobjekt c. Es seien GT_1 und GT_2 globale Transaktionen und LT_3 eine lokale Transaktion auf den entsprechenden Knoten:

$$\begin{aligned} GT_1 &: \ w_1(a)w_1(c) \\ GT_2 &: \ w_2(c)w_2(b) \\ LT_3 &: \ r_3(b)r_3(a) \end{aligned}$$

Betrachtet werden soll die folgende Situation: Im Knoten s_2 hält die globale Transaktion GT_2 die lokale und globale Schreibsperre für c, während in s_1 die lokale und globale Schreibsperre von GT_1 gehalten wird. Weiterhin halte LT_3 die lokale Lesesperre für b. Die Transaktion GT_2 erhält vom GTM die globale Schreibsperre für b, wartet jedoch auf die lokale Sperre, die von LT_3 gehalten wird. LT_3 wartet auf die lokale Sperre für a, die GT_1 erworben hat. GT_1 dagegen wartet auf die Freigabe der globalen Schreibsperre für c durch GT_2. Eine globale Verklemmungssituation liegt vor. □

Die dargestellten Beispiele zeigen, daß es auf verschiedenen Ebenen möglich ist, globale Deadlocks zu erzeugen. Folglich müssen bei der Behandlung dieser Situationen auch alle Ebenen betrachtet werden. Verklemmungssituationen in zentralen und homogenen Systemen sind intensiv untersucht worden und verschiedene Mechanismen, die auf dem Prinzip von "wait-for"-Graphen beruhen, sind vorgeschlagen worden, um Deadlocks zwischen Transaktionen zu erkennen. Ein ähnlicher Ansatz soll auch hier gewählt werden. Da sowohl die lokalen Scheduler als auch die Schedulerkomponente des GTM S2PL-Verfahren zur Synchronisation der zu verarbeitenden Transaktionen benutzen, können leicht globale und lokale "wait-for"-Graphen definiert werden. Ein globaler (lokaler) "wait-for"-Graph (GWFG bzw. LWFG) ist demnach ein gerichteter Graph mit der Menge der globalen (lokalen[1]) Transaktionen als Knotenmenge und enthält genau dann eine Kante $T_i \rightarrow T_j$ zwischen zwei Transaktionsknoten, wenn T_i auf eine globale (lokale) Sperre wartet, die von T_j gehalten wird. Folgende Aussage wird in [BST92] bewiesen:

[1] Eine globale Subtransaktion stellt aus Sicht des CDBMSs eine gewöhnliche lokale Transaktion dar und wird dementsprechend auch im LWFG als solche behandelt.

Lemma 7.1 Es sei GL eine Menge globaler und lokaler Transaktionen, die mindestens zwei globale Transaktionen enthält. Es sei GR die Vereinigung aller LWFG, des GWFG und des WFCG. Ein globaler Deadlock existiert genau dann, wenn GR einen Zyklus enthält. □

In [BST92] wird weiterhin bewiesen, daß die minimale Menge von Graphen, die einen Zyklus in GR erzeugen, zumindest einen LWFG enthält, d.h. es ist nicht möglich, daß allein ein WFCG und ein GWFG zusammen eine globale Verklemmungssituation erzeugen können. Um festzustellen, ob in einem föderierten System eine globale Verklemmungssituation eingetreten ist, muß der GTM folglich auf die verschiedenen lokalen "wait-for"-Graphen zugreifen können. Diese Informationen sind jedoch für das globale Transaktionsmanagement nicht verfügbar, so daß andere Mittel und Wege gefunden werden müssen, um globale Deadlocks entsprechend zu erkennen. In [BST90, BST92] wird mit dem *potentiellen Konfliktgraphen* (PCG – "potential conflict graph") ein Graph vorgestellt, der näherungsweise der Vereinigung der LWFG entspricht. Ebenso wie der "geschätzte" WFG aus der Einleitung dieses Abschnitts kann dieser Ansatz zur Aufdeckung "falscher" Verklemmungssituationen führen, d.h. es werden Verklemmungssituationen erkannt, wenn gar keine vorliegen. Er sichert jedoch, daß alle "richtigen" Deadlocks ebenfalls erkannt werden.

Zur Erläuterung des PCG-Ansatzes ist es nötig, zunächst den Ausführungsstatus einer Transaktion genau zu spezifizieren. Eine Transaktion T_i *ist aktiv im Knoten* s_j, wenn T_i einen Server in s_j hat und der Server gerade eine Operation von T_i ausführt oder die Ausführung einer Operation beendet hat und bereit ist, die nächste Operation von T_i zu empfangen. Ist eine Transaktion nicht aktiv in s_j, so heißt T_i *wartend in* s_j, vorausgesetzt T_i hat einen Server in s_j und zumindest eine Operation von T_i wurde zur Ausführung an den Server übergeben. Ist eine Transaktion aktiv oder wartend in einem Datenbankknoten, so wird sie in dem Knoten *ausgeführt*. Es wird davon ausgegangen, daß eine globale Transaktion in höchstens einem Datenbankknoten in einem wartenden Zustand ist. Da eine Transaktionsoperation erst ausgeführt wird, nachdem die vorangegangene Operation ausgeführt wurde, gilt diese Annahme offensichtlich.

Ein potentieller Konfliktgraph ist dann ein gerichteter Graph mit der Menge der globalen Transaktionen als Knotenmenge, der genau dann ein Kante $GT_i \rightarrow GT_j$ zwischen zwei globalen Transaktionen enthält, wenn es einen Knoten s_k gibt, in dem GT_j aktiv ist und GT_i wartet.

Demnach verändert sich der PCG, sobald sich der Status einer globalen Transaktion ändert. Ist eine Transaktion in s_j in einem Wartezustand, so wartet sie auf die Freigabe einer lokalen Sperre durch das CDBMS. Erhält die Transaktion die Sperre, wechselt der Zustand von "wartend" zu "aktiv". Hat ein globale Transaktion alle Datenbankoperationen beendet und wird die Terminierung der Transaktion eingeleitet, verweilt die Transaktion im aktiven Zustand, bis sie abbricht oder vollständig beendet wurde. Kommt es während der Terminierung zu einer Ausnahmesituation, so daß eine Redo-Transaktion in einem Knoten gestartet werden muß, so fordert die Transaktion die dafür

benötigten lokalen Sperren an und wechselt in dem entsprechenden Knoten in den Wartestatus. Wurde eine globale Transaktion abgebrochen oder in allen Datenbankknoten vollständig beendet, so wird der entsprechende Knoten der Transaktion aus dem PCG entfernt. Folgende Aussage wird in [BST92] bewiesen:

Lemma 7.2 Es sei GG die Vereinigung des PCG, des GWFG und des WFCG. Ist GG azyklisch, so gibt es keine Möglichkeit für eine globale Verklemmungssituation, vorausgesetzt, daß alle lokalen Schedules verklemmungsfrei sind. □

In [BST90, BST92] wird ein ausführlicher Recovery-Manager-Algorithmus (RM) vorgestellt, der alle möglichen globalen Verklemmungssituationen entdeckt und entsprechend behandelt. Dieser Algorithmus setzt voraus, daß jede globale Transaktion GT_i vom GTM beim Start einen eindeutigen Zeitstempel $ts(GT_i)$ erhält.

Der RM behandelt die Anforderung einer lokalen Sperre wie folgt: GT_i fordere auf s_j eine lokale Sperre an. Wurde die Anforderung gestellt, wechselt der Status von GT_i zu "wartend" und der PCG wird entsprechend modifiziert. Der RM setzt ein Timeout-Intervall für GT_i fest und wartet auf die Antwort des Servers. Ist das Timeout-Intervall ohne Antwort des Servers abgelaufen, bildet der GTM den GG und durchsucht ihn nach Zyklen. Ist GG zyklenfrei, so wird ein neues Timeout-Intervall festgelegt und die Anforderung der lokalen Sperre wiederholt. Ansonsten sei $\{GT_{j_1}, \ldots, GT_{j_n}\}$ die Menge aller in s_j aktiven globalen Transaktionen, die im GG in einer Schleife mit T_i erscheinen. Wenn $ts(T_i) < min(ts(GT_{j_1}), \ldots, ts(GT_{j_n}))$ gilt, so wird ein neues Timeout-Intervall für GT_i festgelegt und die Sperranforderung wiederholt, ansonsten wird GT_i in allen Datenbankknoten abgebrochen. Bestätigt der Server dagegen rechtzeitig den Erhalt einer lokalen Sperre, wechselt der Status von GT_i zu "aktiv" und der PCG wird entsprechend modifiziert.

Die Verarbeitung einer globalen "commit"-Operation wird durch den RM folgendermaßen gehandhabt: Es sei SS die Menge der lokalen Datenbankknoten, in denen GT_i ausgeführt wurde. Nach dem Verschicken der "prepare-to-commit"-Operation an alle Knoten in SS setzt der RM ein Timeout-Intervall und wartet auf die Nachrichten der entsprechenden Server. Antwortet mindestens ein Server mit "abort" oder läuft das Timeout-Intervall ab, ohne daß alle Antworten eingetroffen sind, so wird GT_i abgebrochen. Antworten alle Server rechtzeitig mit "ready", so wird eine globale "commit"-Nachricht an alle Knoten in SS gesendet. Der RM setzt ein erneutes Timeout-Intervall und wartet auf die Bestätigungen der vollständigen "commit"-Ausführung der Server. Ist das Timeout-Intervall abgelaufen, so gilt für alle Knoten s_j, die nicht geantwortet haben folgendes: Wurde GT_i in s_j noch nicht wiedergestartet ("restarted"), so wird GT_i in s_j mit einem neuen Zeitstempel wiedergestartet, der Status wechselt zu "wartend" und der PCG wird entsprechend modifiziert. Anschließend wird GG gebildet und auf Zyklen durchsucht. Enthält GG einen Zyklus, so sei $\{GT_{j_1}, \ldots, GT_{j_n}\}$ die Menge aller in s_j aktiven globalen Transaktionen, die im GG in einer Schleife mit T_i erscheinen. Der RM bricht dann eine Transaktion GT_{j_l} ab, für die $ts(T_{j_l}) = max(ts(GT_{j_1}), \ldots, ts(GT_{j_n}))$ gilt, wobei GT_{j_l} keine wiedergestartete Transaktion sein darf. Enthält GG keinen Zyklus,

so wird ein neuer Timeout-Intervall festgelegt und obige Prozedur wiederholt. Bestätigen alle Server innerhalb des Timeout-Intervalls die Vollendung der "commit"-Operation, so wird GT_i aus dem PCG und dem GWFG entfernt, alle gehaltenen Sperren freigegeben und alle Server in den Knoten aus SS gelöscht.

Der angegebene Algorithmus entdeckt alle mögliche globalen Verklemmungen und löst diese entsprechend auf. Keine wiedergestartete Transaktion wird vom RM abgebrochen, dagegen werden die "Restarts" bei erneuten Fehlersituationen solange wiederholt, bis die globale Transaktion ihre "commit"-Operation vollständig beendet hat. Der Algorithmus arbeitet bei den angegebenen Beispielen folgendermaßen:

Im Beispiel 3.4 nehmen wir an, daß das Timeout-Intervall für GT_1 im Knoten s_2 abgelaufen ist. Die Anforderungsprozedur für lokale Sperren überprüft den GG. In diesem Fall enthält der PCG den Zyklus $GT_1 \rightarrow GT_2 \rightarrow GT_1$. Da in s_2 nur GT_2 aktiv ist und GT_1 einen kleineren Zeitstempel hat, wird ein neues Timeout-Intervall festgelegt. Läuft jedoch das Timeout-Intervall von GT_2 in s_1 ab, so wird GT_2 abgebrochen und dadurch die Verklemmung aufgelöst.

Im Beispiel 7.1 bestätigt der Server von GT_1 in s_2 nicht die Vollendung der "commit"-Operation, da ein Fehlerzustand eingetreten ist. Der RM startet den Server mit einem neuen Zeitstempel, setzt den Status von GT_1 in s_2 auf "wartend" und initialisiert das Timeout-Intervall. Die Vereinigung von WFCG und PCG enthält den Zyklus $GT_1 \rightarrow GT_2 \rightarrow GT_1$. GT_2 ist die einzige aktive globale Transaktion in s_2 und wird darum vom RM abgebrochen, so daß GT_1 erfolgreich beendet werden kann.

Im Beispiel 7.2 wartet GT_1 auf eine globale Sperre von GT_2 und der PCG enthält die Kante $GT_2 \rightarrow GT_1$. Folglich enthält GG einen Zyklus. In s_1 ist GT_1 aktiv, während GT_2 wartet. Ist nun das Timeout-Intervall von GT_2 abgelaufen, so wird GT_2 abgebrochen, da diese Transaktion den größeren Zeitstempel hatg GT_1 ist in der Lage, die globale und lokale Sperre für c zu erhalten und kann somit in der Abarbeitung fortfahren.

Die Beispiele zeigen, daß mit Hilfe dieses Algorithmus die globalen Verklemmungssituationen erkannt und vermieden werden können. Aufgrund der Unterschiedlichkeit der betrachteten Situationen scheint dieser Ansatz alle möglichen globalen Deadlocks zu erkennen und geeignet aufzulösen. An dieser Stelle soll die Betrachtung der globalen Verklemmungssituationen beendet werden. Das nachfolgende Kapitel faßt die in diesem und in den vorangegangenen Kapiteln getroffenen Aussagen zusammen.

Kapitel 8

Übersicht der vorgestellten Verfahren

In den vorangegangenen Kapiteln 4 bis 7 sind verschiedene Korrektheitskriterien und eine Reihe von Realisierungsansätzen für die unterschiedlichen Probleme der Transaktionsverwaltung in föderierten DBMSen vorgestellt worden. Innerhalb dieses Kapitels soll ein zusammenfassender Überblick über die vorgestellten Verfahren gegeben und sowohl die einzelnen Ansätze als auch die kompakten Lösungsvorschläge miteinander verglichen werden.

In Kapitel 4 sind zunächst Ansätze zur Sicherung der Serialisierbarkeit globaler Abarbeitungen vorgestellt worden. Mit Blick auf die integrierbaren DBMSe wurde dabei zwischen "labeled" und "unlabeled black boxes" unterschieden. Föderierte Systeme mit "unlabeled black boxes" als teilnehmende Datenbanksysteme sind im Hinblick auf die Designautonomie bedeutend weniger restriktiv als FDBSe mit "labeled black boxes" als Teilnehmer an der Föderation, da erstere lediglich verlangen, daß die CDBMSe lokal die ACID-Eigenschaften der Transaktionen sichern, d.h. von den Schedulern wird angenommen, daß sie serialisier- und rücksetzbare Schedules erzeugen. Durch die Interaktion globaler Subtransaktionen mit lokalen Transaktionen können dabei Probleme für den GTM bei der Sicherung globaler Serialisierbarkeit entstehen, die nicht durch die Ausnutzung lokaler Schedulereigenschaften gelöst werden können. Für solche Systeme wurden die Kriterien CCSR, SSR und HSR vorgestellt, deren Forcierung unter den globalen Transaktionen es dem GTM möglich machte, in abbruch- und fehlerfreien Umgebungen die lokale Serialisierungsreihenfolge der globalen Subtransaktionen zu bestimmen.

> *Die Forcierung von CCSR-, SSR- oder HSR-Schedules sichert die globale Serialisierbarkeit, ohne Einschränkungen der lokalen Autonomie vorzunehmen.*

Mit der OTM und der CTM [GRS91, GRS94] wurden zwei Realisierungsansätze zur Durchsetzung der CCSR-Eigenschaft zwischen den globalen Transaktionen in "unlabeled" Systemumgebungen vorgestellt. Beide Methoden erhalten die lokale Autonomie

insofern, daß sie keine Modifizierung der CDBMSe und der Abarbeitung lokaler Transaktionen fordern. Beide Verfahren gehen lediglich von der minimalen Forderung nach Durchsetzung der ACID-Eigenschaften lokaler Abarbeitungen aus. Die Methoden lösen das globale Serialisierbarkeitsproblem durch das Forcieren direkter Konflikte zwischen den Subtransaktionen einer globalen Transaktion, indem sie von jeder globalen Subtransaktion verlangen, in dem entsprechenden Datenbankknoten eine Ticketoperation auszuführen. Die Effekte indirekter Konflikte, die in Abschnitt 3.3.1 die globale Serialisierbarkeit gefährdeten, spielen bei diesem Ansatz keine Rolle mehr, da sie aufgrund der geforderten lokalen Serialisierbarkeit nicht die Serialisierungsreihenfolgen globaler Subtransaktionen durcheinander bringen können. Die optimistische Ticketmethode ist im Vergleich zum pessimistischen CTM-Ansatz mit Blick auf die lokalen Schedules weniger restriktiv. Eine Validierung der Serialisierungsreihenfolge der globalen Subtransaktionen erfolgt erst zum "commit"-Zeitpunkt mit Hilfe eines globalen Serialisierbarkeitsgraphen (GSG). Da die Ausführung der Ticketoperationen durch den GTM nicht koordiniert sondern lediglich überwacht wird, kann dieser Ansatz zu Restarts globaler Transaktionen führen. Im Gegensatz dazu koordiniert die CTM die Ausführung der Ticketoperationen und verlangt dazu einen sichtbaren "prepared-to-take-a-ticket"-Zustand. Da es sich hierbei um einen simulierten Zustand handelt, ist die Designautonomie der Systeme nicht verletzt, lediglich die Ausführungsautonomie wird durch den zusätzlichen Wartezustand innerhalb einer globalen Subtransaktion eingeschränkt. Da aber andere Ansätze prinzipiell von einer schrittweisen Verarbeitung globaler Transaktionen ausgehen, stellt dieser Zustand eine durchaus akzeptable Einschränkung der Ausführungsautonomie dar.

> *Die OTM und die CTM realisieren CCSR-Schedules durch das Forcieren direkter Ticketkonflikte zwischen den globalen Transaktionen in jedem Knoten und sichern so die globale Serialisierbarkeit.*

Problematischer muß dagegen die Forderung nach einem sichtbaren "prepared-to-commit"-Zustand betrachtet werden. Beide Ansätze versuchen das globale Atomaritäts- und Fehlertoleranzproblem durch den Einsatz eines 2PC-Protokolls zu lösen, welches einen solchen Zustand voraussetzt. Die OTM benötigt diesen Zustand außerdem, um die Validierung der Serialisierungsreihenfolgen durchzuführen. Die Forderung nach einem solchen Zustand beschränkt sowohl die Design- als auch die Ausführungsautonomie lokaler Systeme, da einerseits nicht mehr jedes lokale DBMS einsetzbar ist und es andererseits den lokalen Systemen verboten ist, Subtransaktionen im "prepared"-Zustand abzubrechen. [GRS91, GRS94] argumentieren, daß man einen "prepared"-Zustand mit ähnlichen Eigenschaften simulieren kann, was das Spektrum der einsetzbaren Systeme wieder erweitert. Jedoch ist zweifelhaft, ob ein simulierter "prepared"-Zustand das globale Atomaritäts- und Fehlertoleranzproblem lösen hilft. Da beispielsweise durch den Einsatz eines "wound-wait"-Mechanismus zur lokalen Verklemmungsbehandlung auch Subtransaktionen im simulierten "prepared"-Zustand abgebrochen werden können, sind die in Kapitel 6 betrachteten globalen Transaktionsfehler sowohl bei der OTM als auch bei der CTM möglich. Da sich bei diesem Ansatz voll auf die 2PC-Methode verlassen

wird und keine zusätzlichen Recovery-Mechanismen durch die Server realisiert werden, scheint das Recovery-Problem aus Abschnitt 3.3.2 nicht ausreichend gelöst. Zur Vermeidung globaler Verklemmungen setzen die OTM und die CTM Timeout-Mechanismen zusammen mit einem globalen "wait-for"-Graphen ein. Dieser Ansatz löst das globale Verklemmungsproblem. Zusammenfassend läßt sich feststellen, daß OTM und CTM die in föderierten Umgebungen auftretenden Probleme lösen, wenn Abstriche im Hinblick auf die Ausführungs- und Designautonomie gemacht werden und nur CDBMSe mit systemunterstützten "prepared-to-commit"-Zustand verwendet werden. Will man die Ausführungs- und Designautonomie weiter fassen und akzeptiert auch simulierte "prepared"-Zustände, sind zur Lösung des Atomaritäts- und Fehlertoleranzproblems zusätzliche Recovery-Mechanismen, wie sie beispielsweise bei der 2PCA-Methode vorgestellt wurden, in den Servern nötig.

> *Die OTM und die CTM erfordern sichtbare Transaktionszustände zur Validierung der Ticketoperationen bzw. zur Teilnahme am 2PC-Protokoll und beschränken dadurch die lokale Autonomie.*

Föderationen mit "labeled black boxes" als Teilnehmer gehen dagegen davon aus, daß zusätzliche Eigenschaften lokaler DBMSe dem GTM bekannt sind und von diesem zur Realisierung global serialisierbarer Ausführungen genutzt werden können. Speziell nutzt der GTM dabei Eigenschaften lokaler Schedules, die über die SR- und RC-Eigenschaft hinausgehen. Innerhalb dieser Arbeit wurden globale Realisierungsansätze für folgende lokale Scheduleeigenschaften vorgestellt: StSR, SP-basierte SR, StRC und RG.

> *Lokale Scheduleeigenschaften wie die strenge Serialisierbarkeit, die SP-basierte Serialisierbarkeit, die strenge Rücksetzbarkeit oder die Rigorosität erleichtern die Aufgaben des GTM bei der Realisierung global serialisierbarer Abarbeitungen und stellen in vielen Fällen akzeptable Restriktionen der lokalen Autonomie dar.*

Lokale Scheduler, die die Eigenschaft der strengen Serialisierbarkeit garantieren, sichern, daß in einem Knoten seriell ausgeführte globale Subtransaktionen entsprechend ihrer lokalen Ausführungsreihenfolge serialisiert werden. Diese Eigenschaft wurde vom Knotengraphenalgorithmus und dem altruistischen Sperrverfahren ausgenutzt. Beide Verfahren sind ursprünglich für einfachere Systeme, d.h. Scheduler, die lediglich SR sichern, definiert worden, erwiesen sich dort jedoch als nicht zur Lösung des globalen Serialisierbarkeitsproblems geeignet. In StSR-Umgebungen realisieren sie jedoch das gewünschte Verhalten, d.h. sie vermeiden das Eintreffen von Situationen, die die globale Serialisierbarkeit verletzen.

Der Knotengraphenalgorithmus [BS88] ist ein sehr pessimistischer Ansatz, der indirekte Konflikte zwischen zwei globalen Subtransaktionen annimmt, sobald sie gemeinsam in einem Datenbankknoten arbeiten. Mit Hilfe eines Knotengraphen wird deshalb ein gleichzeitiges Arbeiten zweier globaler Transaktionen in mehr als einem gemeinsamen

Knoten verhindert. Dieser Ansatz basiert auf der Vermeidung von Zyklen im Knotengraphen. Da nicht jeder Zyklus im Knotengraphen zwangsweise einen Zyklus im globalen Serialisierbarkeitsgraphen nach sich zieht, ist dieses Vorgehen restriktiver als nötig und weist eine sehr geringe Parallelität auf. Betrachtet man den Knotengraphenalgorithmus nur im Hinblick auf die globale Serialisierbarkeit, so erfüllt er maximale Anforderungen an Kommunikations- und Ausführungsautonomie. Durch die Forderung nach StSR-Schedulern ist jedoch die Designautonomie der lokalen Komponenten eingeschränkt. Der Knotengraphenalgorithmus sichert die Freiheit der Abarbeitungen von globalen Verklemmungssituationen, da durch die serielle Ausführung globaler Transaktionen das gegenseitige direkte oder indirekte Warten zweier globalen Transaktionen aufeinander unmöglich wird. Der Knotengraphenalgorithmus beschäftigt sich jedoch nicht mit Fragen der Atomarität und Fehlertoleranz der Ausführungen, so daß durch eine Kombination mit Realisierungsvorschlägen aus Kapitel 6 weitere Einschränkungen der Autonomieforderungen oder Abarbeitungseigenschaften vorgenommen werden müssen.

Das Altruistische Sperren ist ebenfalls ein pessimistischer Ansatz, der das überlappende Arbeiten zweier globaler Subtransaktionen in einem Datenbankknoten verhindert. Jedoch werden die einzelnen Knoten nicht für die gesamte Abarbeitungszeit einer globalen Transaktion gesperrt, sondern lediglich für die Ausführungsdauer der jeweiligen Subtransaktion. Dies führt zu einer Erhöhung der Parallelität gegenüber dem vorherigen Ansatz. Durch die Restriktion auf StSR-realisierende lokale Scheduler und das Verwalten eines "wake"-Graphen gelingt es, in fehler- und abbruchfreien Umgebungen globale Serialisierbarkeit zu gewährleisten. Jedoch wird durch die vorzeitige Freigabe von Ergebnissen die Atomarität der globalen Transaktion in Fällen globaler Abbrüche verletzt. Kompensationstechniken sollen in diesem Fall ein Rücksetzen auf einen konsistenten Zustand gewährleisten. Da der [AGS87]-Artikel bis zum Ende dieser Arbeit nicht eingesehen werden konnte, sondern lediglich ein allgemeiner Artikel zum altruistischen Sperren vorlag [SGA89], resultieren die in dieser Arbeit gemachten Ausführungen im wesentlichen aus Charakterisierungen und Referenzen thematisch verwandter Publikationen. Da sich daraus aber nur ein sehr unvollständiges Bild dieses Ansatzes ergibt, soll auf eine weitere Bewertung an dieser Stelle verzichtet werden. Tabelle 8.1 faßt die bisher dargestellten Aspekte zusammen.

> *Der Knotengraphenalgorithmus und das altruistisches Sperren benötigen StSR-Scheduler in allen CDBSen zur Realisierung global serialisierbarer und verklemmungsfreier Abarbeitung, stellen jedoch keine Recovery-Mechanismen zur Verfügung.*

Durch den Einsatz von Schedulern, die die Serialisierungsreihenfolge der auszuführenden Transaktionen mit Hilfe von Serialisierungspunkten innerhalb der Transaktionen festlegen, kann die vorherige Vorgehensweise entscheidend verbessert werden. Im Top-Down-Ansatz von [DE90] und im TSG-Ansatz von [MRB+92b] wurde das Wissen über solche Scheduler zur Sychronisation globaler Subtransaktionen genutzt. Prinzipiell reduzieren solche lokalen Scheduler die serielle Ausführung der globalen Subtransaktionen auf

	OTM	**CTM**	**Knotengraph**	**Altr. Sperren**
Prinzip	Forcierung direkt. Konflikte zwischen GT (Ticketoperation)	wie OTM	serielle Ausführung globaler Transaktionen	serielle Ausführung globaler Subtransaktionen
globales Serialisierbarkeitskriterium	GSR	GSR	GSR	GSR
lokale Synchr.-mechanismen	SR, sichtbarer "prep.-to-commit"-Zustand	SR, sichtbare "prep.-to-commit"-, "prep.-to-take-a-ticket"-Zust.	StSR	StSR
lokale Autonomie	eingeschränkt durch sichtbare Zustände	eingeschränkt durch sichtbare Zustände	eingeschränkt durch StSR	eingeschränkt durch StSR
globale Recovery-Mechanismen	2PC	2PC	keine Angabe	kein atomares "commit", Kompensation bei GT-Abbrüchen
globale Verklemmungsbehandlung	"Timeout"-Mechanismen, "Wait-For"-Graph	wie OTM	bei serieller Ausführ. d. GT u. lok. StSR-Schedules deadlockfrei	bei serieller Ausführ. d. GST u. lok. StSR-Schedules deadlockfrei
verwaltete Strukturen	GTM verwaltet GSG für mit "commit" beendete Transakt.	kein GSG	GTM verwaltet bipartiten Knotengraphen	GTM verwaltet Gefolgegraph und Knotenmarkierungen
Vorteile	keine Restriktion d. lok. Ausführungen, nur geringe lokale Anforderungen	keine globalen Restarts, GTM einfacher als bei OTM	beugt globalen Deadlocks vor	deadlockfrei, höhere Parallelität als Knotengraphenalgorithmus
Nachteile	Verw. d. GSG, globale Restarts möglich, zusätzl. Konfl. zw. GT	"prep-to-take-a-ticket"-Zustand nötig	Verwaltung des Knotengraphen, sehr restriktiv, gering. Parallel.	Verwaltung des Gefolgegraphen, kein atomares "commit" der Subtrans.

Tabelle 8.1: Methoden zur Sicherung globaler Serialisierbarkeit

eine serielle Ausführung jener Teilabschnitte globaler Subtransaktionen, die die Operationen vom Beginn der Subtransaktion bis einschließlich Serialisierungspunkt umfassen. Beim Top-Down-Ansatz legt dabei der GTM die Serialisierungsreihenfolge der globalen Transaktionen fest und reicht sie zusammen mit den einzelnen Subtransaktionen zur Durchsetzung an die "Stub"-Prozesse weiter. Diese setzen die global bestimmte Reihenfolge lokal um, indem sie die Abarbeitung der Serialisierungspunkte der Subtransaktionen individuell kontrollieren. Dazu werden durch die "Stub"-Prozesse in Abhängigkeit vom jeweiligen lokalen Scheduler mitunter Kommunikationsereignisse forciert, was eine gewisse Einschränkung der Kommunikationsautonomie zur Folge hat, die jedoch nicht als gravierend betrachtet wird. Der Top-Down-Ansatz sichert die Freiheit der Abarbeitung von globalen Verklemmungen, da die Serialisierungsreihenfolge vom GTM vor dem Start der Abarbeitung bestimmt wird und so eine globale Transaktion nur auf Sperren von globalen Transaktionen warten kann, die vor ihr serialisiert worden. Eine Betrachtung von lokalen Fehlern und Abbrüchen findet in [DE90] nicht statt, so daß zusätzliche Einschränkungen und Mechanismen zu ihrer Behandlung nötig werden.

Gleiches gilt für den TSG-Ansatz. Dieser reduziert das Problem der Sicherung der globalen Serialisierbarkeit auf die Sicherung der Serialisierbarkeit des Schedules, der alle lokalen Serialisierungspunkte enthält. Dies setzt jedoch voraus, daß sämtliche lokalen Schedulermechanismen in der zentralen GTM-Komponente und nicht wie im vorherigen Ansatz in den Servern bekannt sind. Der TSG-Ansatz verfeinert den ursprünglichen Knotengraphenansatz, indem er Zyklen im Graphen erlaubt, jedoch die Ausführung von in Zyklen involvierten Transaktionen mit zusätzlichen Bedingungen versieht. Obwohl dieser Ansatz die wohl konkretesten Implementierungshinweise gibt und damit den komplexesten Realisierungsvorschlag dieser Arbeit bildet, erscheint sein Einsatz im Hinblick auf die Flexibilität des GTM eher ungeeignet. Da weder Untersuchungen zur Verklemmungsproblematik noch zur Realisierung von Recovery-Mechanismen erfolgen, bleiben viele Fragen über die Umsetzbarkeit und Zweckmäßigkeit dieses Ansatzes offen. Gleiches gilt auch für die drei weiteren Ansätze, die in [MRB+92b] detailliert vorgestellt wurden. Die Vorstellung der einzelnen Synchronisationsmechanismen erfolgt jedoch unter ausführlicher Beobachtung der Leistungsfähigkeit, des Parallelitätsgrades und der Komplexität der einzelnen Ansätze und bieten damit eine gute Grundlage für vergleichende Betrachtungen über die Güte globaler Transaktionsverwaltungen.

> *Der Top-Down- und TSG-Ansatz erhöhen durch die Ausnutzung von Wissen über lokale Serialisierungspunkte die Effizienz des GTM bei der Realisierung globaler Serialisierbarkeit. Der Top-Down-Ansatz sichert zusätzlich Verklemmungsfreiheit, vernachlässigt aber wie der TSG-Ansatz die Betrachtung lokaler Fehler- und Abbruchsituationen.*

Die strenge Rücksetzbarkeit war das nächste untersuchte lokale Korrektheitskriterium. Unter der Voraussetzung lokaler Schedules, die dieses Kriterium erfüllen, in allen an der Föderation beteiligten Datenbanksystemen ließ sich die Durchsetzung der globalen Serialisierbarkeit durch den GTM weiter vereinfachen und auf die Realisierung

serieller globaler "commit"-Prozeduren reduzieren. Zunächst wurden mit dem COCO und dem CORCO [Raz92] zwei lokale Schedulerkomponenten vorgestellt, die lokal eine "commit"-Sortierung entsprechend der StRC-Eigenschaft vornehmen und deren Einsatz in föderierten Umgebungen möglich ist. Da diese Komponenten jedoch an den Einsatz weiterer lokaler und globaler Komponenten gebunden sind, die nicht im einzelnen vorgestellt wurden, soll auf eine zusammenfassende Betrachtung an dieser Stelle verzichtet werden. Eine Darstellung dieser Ansätze erfolgt jedoch in Tabelle 8.2.

Als vollständiger Ansatz, der die StRC-Eigenschaft ausnutzt, wurde die ITM [GRS91, GRS94] vorgestellt. Im Gegensatz zur OTM und CTM sicherte die ITM die globale Serialisierbarkeit nicht durch die Forcierung zusätzlicher Ticketkonflikte, sondern nur durch die Ausnutzung der StRC- bzw. der etwas allgemeineren AESO-Eigenschaft lokaler Schedules. Realisieren alle lokalen Scheduler diese Eigenschaft, so sind durch die lokalen "commit"-Reihenfolgen implizite Ticket- und damit Serialisierungsreihenfolgen gegeben. Die ITM hat gegenüber den anderen Ticketmethoden den Vorteil, daß sie keine zusätzlichen Konflikte benötigt und damit die Parallelität und den Durchsatz globaler Transaktionen nicht zusätzlich einschränkt. Im Vergleich zum Knotengraphenansatz und dem altruistischem Sperren muß erwähnt werden, daß bei der ITM eine beliebige Anzahl globaler Transaktionen gleichzeitig in den einzelnen Datenbankknoten arbeiten darf. In [GRS91, GRS94] wurden außerdem eine ACA OTM und eine ACA CTM vorgestellt, die Verfeinerungen der OTM bzw. CTM für die Fälle darstellen, in denen die lokalen Scheduler zusätzlich die ACA-Eigenschaft der Schedules sichern. Für alle hier aufgeführten Ticketmethoden gelten in puncto Verklemmungs-, Fehler- und Abbruchbehandlung die Aussagen zur OTM und CTM. Ähnliches gilt natürlich auch für die dort getroffenen Aussagen zur Autonomie der lokalen Systeme, jedoch sind die ACA OTM, ACA CTM und ITM im Hinblick auf die Designautonomie bedeutend restriktiver.

> *In lokalen DBMSen, die Schedules mit StRC- (CO-) bzw. AESO-Eigenschaften garantieren, impliziert die "commit"-Reihenfolge die lokale Serialisierungsreihenfolge. Die ITM nutzt diese lokalen Informationen für ein effektives globales Transaktionsmanagement, daß globale Serialisierbarkeit gewährleistet, erfordert aber wie die OTM und die CTM sichtbare Transaktionszustände.*

Die restriktivste, aber auch am besten durch einen GTM zu nutzende lokale Scheduleeigenschaft, die in dieser Arbeit untersucht wurde, ist die Rigorosität lokaler Abarbeitungen. Ein konkreter Realisierungsvorschlag, der sich explizit auf rigorose lokale Ausführungen bezieht, wurde nicht vorgestellt, jedoch gilt die Kombination von rigorosen lokalen Abarbeitungen mit einem 2PC-Protokoll als universelles Modell eines GTM in föderierten Umgebungen und wird in vielen Publikationen zum Vergleich herangezogen. Vielmehr wird eine Restriktion auf rigorose lokale Ausführungen in einer Reihe von Forschungsarbeiten als akzeptable Verletzung der lokalen Autonomieforderung angesehen, da sehr viele kommerzielle Systeme entsprechende Scheduler besitzen. Es kann auch gezeigt werden, daß alle bisher aufgeführten Ansätze vom Einsatz rigoroser lokaler Abarbeitungen profitieren, da dadurch eine Reihe von Restriktionen des globalen

	Top-Down-Ansatz(1)	**TSG-Mechanismus**	**CO** (1)	**CO** (2)
Prinzip	indiv. *Stub-Proz.* kontrollieren das Erreichen d. SP der GT in den einz. CDBMSen	Kontrolle des Einreichens d. GT-SP mittels geeigneter Strukturen	COCO sichert für beliebige Scheduler die CO-Eigenschaft d. Terminierung	CORCO sichert für beliebige Scheduler die CO- und RC-Eigenschaft der Terminierung
globales Serialisierbarkeitskriterium	GSR	GSR	GSR	GSR
lokale Synchr.-mechanismen	O-SR (SP-Ansatz)	SP basierte Serialisierbarkeit	StRC (CO durch COCO)	StRC (CO und RC durch CORCO)
lokale Autonomie	eingeschränkt durch SP-Ansatz	eingeschränkt durch SP-Ansatz	eingeschränkt durch StRC-Forderung	eingeschränkt durch StRC-Forderung
globale Recovery-Mechanismen	keine Angabe	keine Angabe	2PC	2PC
globale Verklemmungs-behandlung	deadlockfrei	keine Angabe	deadlockfrei bei Kombination m. nicht-blockierenden lokalen Schedulern	deadlockfrei bei Kombination m. nicht-blockierenden lokalen Schedulern
verwaltete Strukturen	indiv. *Stub-Proz.* für jedes CDBMS	GTM verwaltet TSG und div. Hilfsstrukturen	COCO verwaltet USG für nicht-beendete Trans.	CORCO verwalt. USG und *wrf*-USG für nicht-beendete Trans.
Vorteile	deadlockfrei, wenige globale Abbrüche, Ans. durch ind. *Stub-Proz.* beliebig ausbaubar	liefert konkr. Impl.-ansatz, ist zu Algorith. mit höherer Parallelität ausbaubar	COCO mit beliebigen RAS o. Schedulern kombinierbar, beeinflußt nicht deren Eigenschaften	CORCO mit beliebigen RAS o. Schedulern kombinierbar, beeinflußt nicht deren Eigenschaften
Nachteile	lok. Scheduling-Mechanismen bestimmen glob. Parallel.-grad	komplex zu verwalt. Strukt., Kenntnis der lok. SP im GTM nötig	Verwaltung d. USG, keine Betrachtung lokaler Trans.	Verwaltung d. USG und *wrf*-USG, keine Betrachtung lokaler Trans.

Tabelle 8.2: Methoden zur Sicherung globaler Serialisierbarkeit

Transaktionsmanagement wegfallen und viele Mechanismen zur Realisierung globaler Serialisierbarkeit einfach überflüssig würden.

> *Die Rigorosität wird in vielen Fällen als akzeptable Einschränkung der lokalen Autonomie in föderierten Systemen angesehen. Durch die Forcierung "commitverzögerter" globaler Transaktionen ist dem GTM in diesen Umgebungen ein einfacher Mechanismus zur Durchsetzung globaler Serialisierbarkeit gegeben.*

Im Kapitel 5 wurde untersucht, inwieweit Alternativen zur globalen Serialisierbarkeit gefunden und in föderierten Umgebungen sinnvoll angewendet werden können. Ziel dieses Kapitels war die Schaffung eines weniger restriktiven Korrektheitskriteriums für die Synchronisation globaler Abarbeitungen, das die Leistungsfähigkeit einer Föderation in puncto Parallelität erhöht, weniger globale Transaktionsabbrüche erfordert und die Autonomie der lokalen Systeme besser als bisher erhält. In diesem Kapitel wurden die theoretischen Ansätze der LSR, 2LSR und QSR erläutert. Eine ausführlichen Vorstellung eines praktischen Realisierungsansatzes erfolgte nicht oder nur teilweise. Zur Entwicklung und Untersuchung der alternativen Ansätze war ein allgemeines Kriterium nötig, das die Korrektheit eines alternativen Ansatzes bestätigt. Mit der "strengen Korrektheit" einer Ausführung wurde so ein allgemeingültiges Kriterium gefunden.

> *In einer Reihe von Anwendungsszenarien ist es aus Sicht der Konsistenzerhaltung einer Abarbeitung nicht nötig, globale Serialisierbarkeit zu fordern.*

Der erste untersuchte alternative Ansatz war die lokale Serialisierbarkeit globaler Schedules, die lediglich forderte, daß alle lokalen Projektionen eines globalen Schedules serialisierbar sein müssen. Globale Abarbeitungen, die die LSR-Eigenschaft aufweisen, sind streng korrekt, wenn ausschließlich lokale Konsistenzbedingungen gesichert werden müssen und alle verarbeiteten Transaktionen LDP- oder NVD-Eigenschaften aufweisen. Die NVD-Transaktionen sind jedoch eine echte Teilmenge der LDP-Transaktionen, so daß die ausschließliche Realisierung von NVD-Transaktionen restriktiver als nötig ist. Da lediglich die Serialisierbarkeit der einzelnen lokalen Projektionen des globalen Schedules gefordert wird, kann die Wahrung von globalen Konsistenzbedingungen durch einen LSR-Schedule nicht gesichert werden. Demnach können keine globalen Ausführungsbedingungen gestellt werden und der GTM muß somit keine Synchronisationsfunktionen umsetzen. Dadurch erhöht sich die Parallelität der Ausführungen und der Durchsatz an globalen Transaktionen. Gleichzeitig bleibt die Autonomie der lokalen Systeme uneingeschränkt erhalten. Es werden keine Anforderungen an die lokalen Scheduler und Ausführungen gestellt. Jede lokale Transaktion ist durch ihre Beschränkung auf einen Knoten von sich aus LDP bzw. NVD. Für globale Transaktionen ist diese Eigenschaft durch den Anwender oder den GTM sicherzustellen, was die Anwendbarkeit dieses Ansatzes jedoch recht stark eingrenzt.

	ITM	**ACA OTM**	**"commit"-Graph**	**2PCA-Methode**
Prinzip	Ausnutzung lok. Schedule-eigenschaften	wie OTM und Ausnutzung lok. Schedule-eigenschaften	Sicherung der Atomarität glob. Transaktionen durch Wiederholung lokaler Subtransaktionen	Sicherung der Atomarität glob. Transaktionen durch Wiederholung lokaler Subtransaktionen
globales Serialisierbarkeitskriterium	GSR	GSR	GSR	GSR (GVSR) (als Single-step-certifier: QSR)
lokale Synchr.-mechanismen	SR, AESO sichtb. "prep.-to-commit"-Zustand	SR, ACA, sichtbarer "prep.-to-commit"-Zustand	RG	RG
lokale Autonomie	eingeschränkt durch sichtbare Zustände und AESO-Forderung	eingeschränkt durch sichtbare Zustände und ACA-Forderung	eingeschränkt durch RG und Beschränkung des Zugriffs	eingeschränkt durch RG und Beschränkung des Zugriffs
globale Recovery-Mechanismen	2PC	2PC	Redo-Ansatz	Redo-Ansatz
globale Verklemmungs-behandlung	wie OTM	wie OTM	Verwaltung des PCG sichert Auflösung von glob. Verklemmungssituationen	Timeout-Mechanismus wird angeraten
verwaltete Strukturen	kein GSG	kein GSG	CG, WFCG, PCG	"Alive-Time"-Intervalle, Seriennummern f. Transaktionen
Vorteile	keine Ticket-konlikte nötig	kein GSG, Ausnutzung lok. Schedule-eigenschaften	verlangt keine 2PC-fähigen lokalen Interfaces, löst alle aufgef. TV-Probleme	verlangt keine 2PC-fähigen lokalen Interfaces, löst GST- und AFT-Problem
Nachteile	Restriktion der lokalen Synch.-mechanismen	globale Restarts möglich	Restriktion des Zugriffs durch Unterteilung in glob./lok. modifizierbare Daten	Restriktion des Zugriffs durch Unterteilung in glob./lok. modifizierbare Daten

Tabelle 8.3: Methoden zur Sicherung globaler Serialisierbarkeit

Die lokale Serialisierbarkeit globaler Schedules sichert die Wahrung aller lokalen Integritätsbedingungen, ohne die lokale Autonomie der CDBMSe zu beschränken.

Bei der Realisierung zweistufig serialisierbarer globaler Abarbeitungen wird ein etwas anderer Ansatz gewählt. Die zweistufige Serialisierbarkeit verlangt neben der Serialisierbarkeit aller lokalen Projektionen des globalen Schedules auch die Serialisierbarkeit der Projektion auf alle globalen Transaktionsoperationen. Zur Umsetzung von Integritätsbedingungen, die mehrere Knoten umfassen, werden die Datenobjekte jedes Datenbankknotens in lokale und globale Objekte unterteilt. Entsprechend der involvierten Datenobjekte werden dann lokale, globale und gemischte Integritätsbedingungen unterschieden. Zur Sicherung der Realisierbarkeit globaler Integritätsbedingungen wird den lokalen Transaktionen verboten, globale Datenobjekte zu schreiben, da diese in globale Integritätsbedingungen involviert sein können. Betrachtet man diesen Ansatz im Hinblick auf die Wahrung der lokalen Autonomie, so stellt die Unterteilung der Daten in zwei disjunkte Mengen eine Restriktion der lokalen Designautonomie dar. Das Verbot des Schreibens globaler Datenobjekte durch lokale Transaktionen schränkt außerdem die lokale Ausführungsautonomie ein. Unter diesen Einschränkungen wird gezeigt, daß die 2LSR-Eigenschaft globaler Schedules strenge Korrektheit realisiert, wenn alle Transaktionen der LDP- und GDP-Anforderung genügen. Die 2LSR-Schedules sind eine echte Untermenge der LSR-Schedules, d.h. jeder 2LSR-Schedule ist auch LSR. Die 2LSR-Schedules haben jedoch den Vorteil, daß sie im Gegensatz zu den LSR-Schedules globale Integritätsbedingungen unterstützen. Die geforderten Restriktionen dienen ausschließlich diesem Zweck und nicht der Serialisierbarkeit der Projektion auf die globalen Transaktionen. Lockert man die Anforderungen an die Realisierung globaler und gemischter Bedingungen, so können auch die Restriktionen der lokalen Autonomie gelockert werden. Die Serialisierbarkeit der Projektion auf globale Transaktionen ist durch einen einfachen Standard-Synchronisationsmechanismus im GTM realisierbar, da indirekte Konflikte bei diesem Ansatz keine Rolle spielen.

Die zweistufige Serialisierbarkeit ist in der Lage, neben lokalen auch globale und gemischte Integritätsbedingungen zu erhalten, wenn die einzelnen Transaktionen nur auf bestimmte Datenmengen zugreifen dürfen.

Von ähnlichen Voraussetzungen ging auch die Quasiserialisierbarkeit aus. Die Menge der QSR-Schedules ist eine echte Untermenge der 2LSR-Schedules, d.h. alle zuvor getroffenen Aussagen gelten auch für die Quasiserialisierbarkeit. Dieser Ansatz kann als Vorgänger des 2LSR-Ansatzes angesehen werden, hatte als solcher aber nicht das Ziel, globale oder gemischte Integritätbedingungen zu unterstützen, so daß in der ursprünglichen Definition keine Restriktionen der lokalen Autonomie vorgenommen wurden. Vielmehr war es zur Sicherung der strengen Korrektheit ausreichend, wenn neben der QSR-Eigenschaft des globalen Schedules alle Transaktionen die NVD-Eigenschaft erfüllten.

Alle drei alternativen Ansätze sind lediglich als theoretische Synchronisationskriterien vorgestellt wurden, die detaillierte Betrachtung eines praktischen Realisierungsansatzes für eines oder mehrere Kriterien erfolgte nicht. Jedoch kann der Top-Down-Ansatz zur Realisierung global serialisierbarer Ausführungen aus [DE90] durch eine geringfügige Modifikation der Einreichungsstrategie der "Stub-Prozesse" ebenso zur Realisierung von quasiserialisierbaren globalen Ausführung verwendet werden. Die dafür von [DE90] vorgeschlagene Strategie scheint jedoch zu restriktiv zu sein, da sie eventuell zur Verfügung stehendes Wissen über lokale Scheduler nicht ausnutzt und eine serielle Einreichung der globalen Subtransaktionen in der Reihenfolge O vorschlägt. Sie realisiert damit quasiserielle Schedules. Ein weiterer Top-Down-Ansatz, der die zweistufige Serialisierbarkeit globaler Ausführungen sichert, wird in [OAB94] vorgestellt. Er arbeitet unter ähnlichen Voraussetzungen wie der [DE90]-Ansatz, gestattet aber wesentlich mehr lokale Autonomie. Die wesentlichen Unterschiede und Merkmale dieses Ansatzes sind im Abschnitt 5.2 erläutert wurden. Dieser Ansatz zeichnet sich vor allem dadurch aus, daß er eine bedeutend größere Bandbreite an lokalen Schedulermechanismen handhabbar macht und einen wesentlich höheren Parallelitätsgrad erreicht, da in den Einreichungsstrategien nur globale Subtransaktionen mit direkten Konflikten untereinander betrachtet werden.

Prinzipiell kann man davon ausgehen, daß zur praktischen Umsetzung eines der vorgestellten alternativen Synchronisationsansätze in einem föderierten Datenbanksystem im Hinblick auf die Verklemmungs- und Fehler- bzw. Abbruchbehandlung zusätzliche Restriktionen bei den lokalen Komponenten durchgeführt werden müssen. In Tabelle 8.4 bzw. 8.5 werden die zuvor getroffenen Aussagen nochmals dargestellt.

Das Kapitel 6 beschäftigte sich mit dem Problem der Fehler- und Abbruchsicherheit föderierter Transaktionsverwaltungen. Einleitend wurde dabei festgestellt, daß die Atomarität globaler Transaktionen in föderierten Umgebungen nicht erreicht werden kann, wenn die lokale Autonomie uneingeschränkt bleibt.

> *Die Realisierung von atomaren und fehler- und abbruchtoleranten globalen Abarbeitungen im Sinn der klassischen ACID-Eigenschaften ist ohne eine Restriktion der lokalen Autonomie nicht möglich.*

Zunächst wurde mit dem 2PC-Protokoll das bekannteste atomare "commit"-Protokoll aus dem Bereich der verteilten Datenbanksysteme vorgestellt. Dieses Protokoll garantierte ein sicheres Handhaben der Situationen, die als globale Transaktionsfehler eingeführt wurden, und löste somit das globale Atomaritäts- und Fehlertoleranzproblem. Problematisch war dabei die Forderung nach einen sichtbaren "prepared-to-commit"-Zustand, die sowohl die lokale Design- als auch die lokale Ausführungsautonomie einschränkt. Zwar bietet die Simulation von solchen Zuständen die Möglichkeit, die Restriktionen der Designautonomie wieder zu lockern, jedoch bieten die simulierten Zustände nicht die gleiche Sicherheit, wie vom System zur Verfügung gestellte Zustände. Unilaterale Abbrüche in simulierten "prepared"-Zuständen sind möglich, wie schon bei den Ticketmethoden argumentiert wurde, und können durch das Protokoll nicht gehandhabt werden. Lokale Fehlersituationen in Datenbankknoten mit simulierten "prepared"-Zuständen können ebenfalls nicht behandelt werden. Aufgrund der genannten Mängel

	allg. LSR-Ansatz	**allg. 2LSR-Ansatz**	**allg. QSR-Ansatz**	**Top-Down-Ansatz** (2)
Prinzip	LDP- oder NVD-Transaktionen sichern strenge Korrektheit	Transaktionen mit GDP- und LDP-Eigenschaften sichern strenge Korrektheit	NVD-Transaktionen sichern strenge Korrektheit	indiv. Server kontrollieren das Erreichen der Ausführungsbedingung für GST aus dem OCDAG
globales Serialisierbarkeitskriterium	LSR	2LSR	QSR	2LSR
lokale Synchr.-mechanismen	SR	SR	SR	SR
lokale Autonomie	nicht eingeschränkt	eingeschränkt durch Verbot der Modifikation globaler Daten durch LT	nicht eingeschränkt	eingeschränkt durch Verbot der Modifikation globaler Daten durch LT
globale Recovery-Mechanismen	nur theoretische Ansätze – verwaltete Strukturen und benutzte Mechanismen sind vom gewählten Implementierungsansatz abhängig			keine Angabe
globale Verklemmungs-behandlung				deadlockfrei
verwaltete Strukturen				indiv. Server für jedes CDBMS u. lokale OCDAG
Vorteile	keine Restriktionen an lokale Scheduler u. LT, keine Unterteilung der Daten	lokale, globale und gemischte Integritäts-bedingungen, keine Restriktion lokaler Scheduler	keine Restriktionen an lokale Scheduler u. LT, keine Unterteilung der Daten	deadlockfrei, wenige globale Abbrüche, Ans. durch indiv. Server beliebig ausbaubar keine Restriktion lokaler Scheduler
Nachteile	keine glob. Integritätsbedingungen, nur GT mit LDP- oder NVD-Eigenschaft erlaubt, begrenzte Einsetzbarkeit	Unterteilung in glob. und lok. Daten, Restriktion des Zugriffs von lokalen Transaktionen, begrenzte Einsetzbarkeit	keine glob. Integritätsbedingungen, nur GT mit NVD-Eigenschaft erlaubt	lok. Scheduling-Mechanismen bestimmen glob. Parallel.-grad, Verwaltung d. OCDAG

Tabelle 8.4: Methoden zur Sicherung alternativer globaler Korrektheitskriterien

	Top-Down-Ansatz (3)	**O2PC-Protokoll**	**allg. ONT/ MLT-Ansatz**	
Prinzip	indiv. *Stub-Proz.* kontrollieren die vollständige Abarbeitung der GST in den einz. CDBMSen	semant. Rücksetzen abgeschlossener glob. Subtrans. bei glob. Transaktionsfehlern sichert "semant. Atomarität"	mehrschichtige Schedules durch unterschiedliche Abstr.-ebenen, ebenenweise semant. SR und RC	
globales Serialisierbarkeitskriterium	QSR	semant. SR	komplette SR	
lokale Synchr.-mechanismen	O-QSR (SR mit serieller Ausführung d. GST in d. Reihenfolge O)	S2PL (SR + ST)	SR auf L_0	
lokale Autonomie	nicht eingeschränkt	eingeschränkt durch S2PL-Scheduler	teilweise eingeschränkt, von konkreten Ansätzen abhängig	
globale Recovery-Mechanismen	keine Angabe	Compensate-Ansatz	Compensate-Ansatz, Komp. auf verschied. Abstr.-ebenen	
globale Verklemmungsbehandlung	deadlockfrei	keine Angabe	von konkreten Ansätzen abhängig	
verwaltete Strukturen	indiv. *Stub-Proz.* für jedes CDBMS	Knotenmarkierungen beim O2PC/P1	ebenenspez. Konfliktrelationen	
Vorteile	deadlockfrei, wenige globale Abbrüche, Ans. durch ind. *Stub-Proz.* beliebig ausbaubar, keine Restriktion lokaler Scheduler	Alternative zu 2PC-Protokoll, frühe Sperrfreigabe, höhere Parallelität	Ausnutzung semant. Aspekte bei Synchr. und Fehlerbehandlung, Steigerung der Parallelität, Lock. d. klass. ACID-Eigensch.	
Nachteile	keine glob. Integritätsbedingungen, nur GT mit NVD-Eigenschaft erlaubt	LT können glob. inkons. Zustände lesen, keine Atomarität der GT	komplexe zu verwalt. Strukt., Abschwächung d. Korrektheit	

Tabelle 8.5: Methoden zur Sicherung alternativer globaler Korrektheitskriterien

wird die Anwendbarkeit eines 2PC-Protokolls in föderierten Umgebungen in vielen Forschungsarbeiten in Frage gestellt.

> *Das 2PC-Protokoll sichert Atomarität und Fehlertoleranz im Sinne der ACID-Eigenschaften, setzt dazu aber einen systemunterstützten "prepared"-Zustand voraus und schränkt damit die lokale Autonomie entscheidend ein.*

Der allgemeine Redo-Ansatz wurde im Kapitel 6 als erste Alternative zum 2PC-Protokoll vorgestellt. Er forciert ein 2PC-Protokoll, in dem nicht die CDBMSe sondern die Server als Teilnehmer fungieren. Bei diesem Ansatz werden globale Transaktionsfehler durch ein wiederholtes Ausführen der "write"-Operationen der Subtransaktionen behoben, die während der "commit"-Prozedur abgebrochen wurden. Zur Lösung des dabei eventuell auftretenden globalen Atomaritäts- und Fehlertoleranzproblems wurde das Kriterium der M-Serialisierbarkeit eingeführt. Um dieses Kriterium durchzusetzen, war eine ganze Reihe lokaler und globaler Restriktionen zu vollziehen. Zunächst müssen alle lokalen Scheduler StRC- und ACA-Eigenschaften sichern und die Projektion des globalen Schedules auf die globalen Transaktionsoperationen muß rigoros sein. Außerdem werden Zugriffsbeschränkungen für globale und lokale Transaktionen formuliert. Je nach Schärfe der definierten Zugriffsbeschränkungen sind Lockerungen der Anforderungen an die lokalen Scheduler möglich. Trotzdem sind zur Sicherung der M-Serialisierbarkeit erhebliche Einschränkungen der lokalen Design- und Ausführungsautonomie nötig. Es wurde gezeigt, daß die Mechanismen zur Sicherung M-serialisierbarer lokaler Abarbeitungen ohne weitere Einschränkungen mit den Korrektheitskriterien LSR und 2LSR verbunden werden können. Um globale Serialisierbarkeit sicherzustellen, reicht die M-Serialisierbarkeit jedoch nicht aus. Zusätzliche Mechanismen, wie eine völlige Entkopplung lokaler und globaler Daten oder die Forcierung einheitlicher "commit"-Reihenfolgen sind dazu nötig. Mit dem "Commit"-Graphen-Ansatz und der 2PCA-Methode wurden zwei Lösungen für die föderierten Transaktionsverwaltung vorgestellt, die sich sowohl der Betrachtung von Synchronisations- als auch Recovery-Aspekten widmen.

Der "Commit"-Graphen-Ansatz [BST90, BST92] ist eine graphenbasierte Lösungsmöglichkeit der Probleme beim föderierten Transaktionsmanagement. Da dabei die Rigorosität aller lokalen Schedules gefordert wird, ist er restriktiver als der oben dargestellte allgemeine Ansatz. Die geforderte Rigorosität der Projektion des globalen Schedules auf die globalen Transaktionsoperationen wird durch einen globalen S2PL-Scheduler sichergestellt, der das gleiche Sperrgranulat wie die lokalen S2PL-Scheduler verwendet. Zur Lösung des Atomaritäts- und Fehlertoleranzproblems werden, dem allgemeinen Ansatz folgend, die Datenobjekte in disjunkte Klassen global und lokal modifizierbarer Daten unterteilt und Zugriffsbeschränkungen aufgestellt. Zur Lösung des globalen Serialisierbarkeitsproblems beim Redo-Ansatz (vergleiche Beispiel 6.1) wird eine in allen Knoten einheitliche "commit"-Reihenfolge forciert. Dazu werden der "commit"-Graph und der "wait-for-commit"-Graph eingeführt. Auch das Problem der Verklemmungsbehandlung wird in diesem Ansatz untersucht und mit Hilfe eines "Potential-Conflict"-Graphen kombiniert mit Timeout-Mechanismen gelöst. Der

dabei gewählte Ansatz ist restriktiver als nötig, beugt aber effizient allen möglichen globalen Verklemmungssituationen vor.

Die 2PCA-Methode [WV90, VW92] geht ebenfalls von rigorosen lokalen Schedulern aus. Außerdem wird lokal die Durchsetzung der DLU-Eigenschaft gefordert. Um diese effektiv umzusetzen, erscheint eine Unterteilung der Datenobjekte in lokal und global modifizierbarer Daten und das Aufstellen von Zugriffsbeschränkungen unumgänglich. Die 2PCA-Methode ist ein zweistufiges Verfahren. Die erste Stufe führt eine Zertifikation zum "prepare"-Zeitpunkt durch und sichert allein (als "single step certifier") die Quasi-(View-)Serialisierbarkeit. Die zweite Stufe führt eine Zertifikation zum "commit"-Zeitpunkt durch und sichert zusammen mit Stufe 1 (als "two step certifier") die globale Viewserialisierbarkeit. Die erste Stufe löst das Atomaritäts- und Fehlertoleranzproblem, indem sichergestellt wird, daß eine spätere Wiederholung einer abgebrochenen Subtransaktion die Sicht dieser Transaktion nicht ändert. Die dabei verwendeten Korrektheitkriterien forcieren die Konfliktfreiheit von globalen Subtransaktionen im "prepared"-Zustand und damit indirekt die Rigorosität der Projektion des globalen Schedules auf die globalen Transaktionsoperationen. Die zweite Stufe löst das globale Serialisierbarkeitsproblem aus Beispiel 6.1, indem eine einheitliche "commit"-Reihenfolge globaler Transaktionen in allen beteiligten CDBMSen gesichert wird. Zur Lösung der globalen Verklemmungsproblematik wird in [VW92] die Anwendung von Timeout-Mechanismen angeraten.

Sowohl der "Commit"-Graphen-Ansatz als auch die 2PCA-Methode scheinen alle Probleme einer föderierten Transaktionsverwaltung zu lösen, akzeptieren dafür aber gleich eine ganze Reihe von Restriktionen der lokalen Autonomie. Im Hinblick auf die lokalen Autonomieanforderungen, konnten keine Unterschiede entdeckt werden, beide Verfahren sind trotz unterschiedlich formulierten Anforderungen scheinbar gleich restriktiv. Es sieht jedoch so aus, als ob die 2PCA-Methode eine größere Bandbreite globaler Schedules zuläßt. Dies ergibt sich zum einen am global forcierten Korrektheitskriterium VSR im Gegensatz zur CSR beim "Commit"-Graphen-Ansatz, andererseits arbeitet der für die 2PCA-Methode vorgeschlagene Mechanismus zur Durchsetzung einer einheitlichen "commit"-Reihenfolge flexibler. Jedoch sind für derartige Betrachtungen sicherlich umfassendere Untersuchungen nötig.

> *Der "Commit"-Graphen-Ansatz und die 2PCA-Methode realisieren ACID-Eigenschaften für globale Transaktionen, fordern dazu aber neben der Rigorosität aller lokalen Schedules auch Zugriffsbeschränkungen für lokale und globale Transaktionen.*

Nach den Redo-Ansätzen wurde der Retry-Ansatz vorgestellt. Dieser Ansatz löst das globale Atomaritäts- und Fehlertoleranzproblem durch die vollständige Wiederholung abgebrochener Subtransaktionen in globalen Transaktionsfehlerfällen. Im Unterschied zu den vorangegangenen Ansätzen ist es dabei erlaubt, daß die Subtransaktion auf in der Zwischenzeit modifizierte Datenobjekte zugreift. Um dennoch eine konsistente Abarbeitung der globalen Transaktion zu sichern, muß die NVD-Eigenschaft zwischen den Subtransaktionen einer globalen Transaktion bestehen. Außerdem müssen wesentliche

Zustände der Originalabarbeitung der globalen Transaktion der zu wiederholenden Subtransaktion zur Verfügung gestellt werden. Zur Kombination dieses Ansatzes mit den globalen Korrektheitskriterien aus den Kapiteln 4 und 5 gelten dieselben Aussagen, wie für die Redo-Ansätze. Da keine konkrete Umsetzung eines Retry-Ansatzes untersucht wurde, sind weitere Aussagen über die Restriktivität und Leistungsfähigkeit nicht möglich.

Die Kompensation abgeschlossener globaler Subtransaktionen nach globalen Transaktionsfehler bildet die Basis der Compensate-Ansätze, die als letztes im Kapitel 6 vorgestellt wurden. Dabei wurden die Effekte bereits mit "commit" beendeter Subtransaktionen einer globalen Transaktion im Falle eines globalen Abbruchs nach semantischen Gesichtspunkten zurückgesetzt. Da es dabei möglich ist, daß lokale Transaktionen Effekte nicht atomar abgearbeiteter Transaktionen sehen, wurde die standardmäßige Atomaritätsforderung gelockert und die semantische Atomarität als umzusetzende Transaktionseigenschaft globaler Transaktionen eingeführt. Mit dem O2PC-Protokoll und dem allgemeinen ONT/MLT-Ansatz wurden zwei Verfahren erläutert, die auf dem Einsatz von Kompensationstechniken beruhen.

Die Lockerung der klassischen ACID-Eigenschaften bildet die Grundlage der Ansätze, die auf der Anwendung von Kompensationstechniken beruhen.

Das O2PC-Protokoll [LKS91b] ist eine Modifikation des verteilten 2PL-Verfahrens, einer Kombination lokaler 2PL-Scheduler mit einem 2PC-Protokoll. Das optimistische 2PC-Protokoll beendet im Gegensatz zum ursprünglichen Ansatz globale Subtransaktionen, sobald die "prepare"-Nachricht im Server eingegangen ist. Nach einer globalen Abbruchentscheidung ist es somit nötig, alle beendeten Subtransaktionen zurückzusetzen. Da aber die Ergebnisse der Subtransaktionen bereits sichtbar waren, kann nur eine semantische Kompensation der Subtransaktion erfolgen. Für diesen Vorgang wird die IR-Eigenschaft gefordert, die sichert, daß jede globale Transaktion nur global konsistente Datenbankzustände sieht. Ein Markierungsalgorithmus sichert die IR-Eigenschaft durch das Vermeiden "regulärer Zyklen" im globalen Serialisierbarkeitsgraphen. Das vorgestellte Protokoll löst das globale Atomaritäts- und Fehlertoleranzproblem durch das vorzeitige Beenden der Subtransaktionen. Müssen diese jedoch kompensiert werden, tritt aus globaler Sicht ein modifiziertes Atomaritätsproblem und Serialisierbarkeitsproblem auf. Durch die Lockerung der globalen Transaktionseigenschaften sind diese Probleme zwar nicht gelöst, stellen aber aus Sicht des globalen Transaktionmanagements auch keine fehlerhaften Bearbeitungen mehr dar. Das O2PC-Protokoll ist eine Alternative zum 2PC-Protokoll, die keine sichtbaren Zustände erfordert. Sie ist damit weniger restriktiv und gewährleistet mehr lokale Design- und Ausführungsautonomie. Das O2PC-Protokoll muß in einer föderierten Transaktionverwaltung mit Mechanismen zur Lösung des globalen Serialisierbarkeits- und Verklemmungsproblems kombiniert werden. Die Art dieser Mechanismen bestimmen letztendlich die gewährleisteten globalen Eigenschaften und den Grad an lokaler Autonomie.

Zum Abschluß soll noch kurz der ONT/MLT-Ansatz genannt werden, der ebenfalls in Kapitel 6 vorgestellt wurde. Dieser Ansatz verbindet die Eigenschaften des mehrschichtigen Transaktionsmanagement für ONT/MLT mit dem Anforderungen des föderierten Transaktionsmanagements. Das MLT-Modell wurde allgemein vorgestellt und einige konkrete Vorschläge zum Einsatz in einem FDBS benannt. Das globale Serialisierbarkeitsproblem wird dabei dadurch gelöst, daß die föderierte Transaktionsverwaltung semantische Serialisierbarkeit sicherstellte. Die komplette Serialisierbarkeit der mehrschichtigen Schedules soll die Probleme bei der Realisierung von Fehler- und Abbruchtoleranz beheben. Da innerhalb dieser Arbeit kein Realisierungsansatz eines föderierten Transaktionsmangement ausführlicher vorgestellt wurde, sind auch keine detaillierten Aussagen zur Wahrung der Autonomie oder Durchsetzung globaler Verklemmungsfreiheit möglich. Es hängt letztendlich von der konkreten Umsetzung ab, inwieweit die Autonomie eines lokalen Systems in der Föderation beschnitten wird. Einige Vorschläge forderten bespielsweise, daß die lokalen Systeme die lokalen Ausführungskonflikte propagieren, um sie höheren Ebenen nutzbar zu machen. Andere Ansätze gingen davon aus, daß die lokalen Systeme in der Lage sind, zwischen globalen Subtransaktionen und lokalen Transaktionen zu unterscheiden, um in L_0 ein 2PL-Verfahren mit "retained lock" durchzusetzen. Beides schränkt die Autonomie der lokalen Systeme ein.

> *Der Einsatz alternativer Transaktionsmodelle kann in vielen Szenarien die Probleme der föderierten Transaktionsverwaltung lösen. Modelle, wie die mehrschichtigen Transaktionen, lockern durch Ausnutzung semantischer Aspekte die klassischen ACID-Eigenschaften für globale und lokale Transaktionen und sind so in der Lage Serialisier- und Rücksetzbarkeit aus semantischer Sicht zu realisieren.*

Im Kapitel 7 wurden Mechanismen zur Behandlung globaler Verklemmungssituationen vorgestellt. Der dabei ausführlich dargestellte "Potentail-Conflict"-Graph-Ansatz ist in dieser Zusammenfassung bereits im Zuge der Beurteilung des "Commit"-Graphen-Ansatzes erwähnt worden. Der PCG-Ansatz beschreibt eine effiziente Methode zur Entdeckung und Behandlung globaler Verklemmungssituationen, die den lokalen Systemen keine zusätzlichen Restriktionen im Hinblick auf deren Autonomie aufzwingt. Jedoch ist dieser Ansatz sehr restriktiv im Hinblick auf die ausführbaren globalen Schedules, da er mitunter Verklemmungssituationen erkennt und globale Transaktionen abbricht, wenn gar keine Deadlocks vorliegen. Insofern ist zu überprüfen, inwieweit dieser Ansatz noch verfeinert werden kann.

Aus den vorangegangenen Ausführungen läßt sich eines erkennen: Die Schaffung eines föderierten Transaktionsmanagements, das die ACID-Eigenschaften für globale und lokale Transaktionen sichert, ist unter vollständiger Wahrung der lokalen Autonomie der an der Föderation partizipierenden Systeme **nicht** möglich. Zur Umsetzung einer vollständigen Lösung für die Transaktionsverwaltung in einem föderierten Datenbanksystem ist entweder eine Einschränkung der lokalen Autonomie oder eine Abschwächung des zu garantierenden globalen Korrektheitskriteriums oder gar beides vorzunehmen. Dabei sind die durch eine Föderation zu realisierenden Eigenschaften wesentliche Kriterien

für die Wahl eines vollständigen Lösungsansatzes oder die Kombination aus Teillösungen. Die Gegenüberstellung der wesentlichen Eigenschaften der einzelnen Ansätze, die innerhalb dieses Kapitels versucht wurde, soll dabei als Grundlage für die Betrachtung und Beurteilung verschiedener Konfigurationen dienen und somit eine Entscheidung für den einen oder anderen Ansatz erleichtern. Die Bandbreite der realisierbaren Systeme ist dabei recht groß:

Legt man beispielsweise in einer Föderation das Hauptaugenmerk auf die Realisierung der ACID-Eigenschaften globaler Transaktionen, so sind dazu zunächst Synchronisationsmechanismen zu wählen, die global serialisierbare Abarbeitungen realisieren. Wünscht man sich gleichzeitig ein Maximum an lokaler Autonomie und möchte die integrierbaren lokalen Synchronisationskomponenten nicht einschränken, so bilden die OTM oder die CTM eine gute Grundlage für ein föderiertes Transaktionsmanagement.

Realisieren dagegen alle zu föderierenden CDBMSe zusätzliche Scheduleeigenschaften, wie strenge Serialisierbarkeit, SP-basierte Serialisierbarkeit oder strenge Rücksetzbarkeit, so lassen sich diese Eigenschaften durch die Verwendung der dafür vorgestellten globalen Synchronisationsmechanismen nutzen, um ein effizientes und effektives föderierten Transaktionsmanagement zu garantieren.

Hat man letztendlich nur die Absicht, lokale Datenbanksysteme mit rigorosen Schedulern in einer Föderation zu verbinden, so ist die Durchsetzung der globalen Serialisierbarkeit sehr viel leichter zu erreichen. Eine einfache "commit"-Verzögerung bis an das Ende der Abarbeitung aller Subtransaktionen einer globalen Transaktion ist ausreichend, um globale Serialisierbarkeit zu gewährleisten.

Ist in diesen Szenarien der Einsatz eines 2PC-Protokoll akzeptabel, d.h. die CDBMSe stellen einen sichtbaren "prepared"-Zustand zur Verfügung, dann sind keine zusätzlichen Restriktionen zur Lösung des Recovery-Problems nötig. Lehnt man den Einsatz eines 2PC-Protokolls dagegen ab, so scheint die Lösung des Recovery-Problems den gravierenden Engpaß bei der Realisierung globaler Schedules zu bilden, die die ACID-Eigenschaften der globalen Transaktionen garantieren. Der "Commit"-Graphen-Ansatz und die 2PCA-Methode haben gezeigt, daß in diesem Fall erhebliche Restriktionen der lokalen und globalen Scheduler und Beschränkungen des Zugriffs der einzelnen Transaktionen vorzunehmen waren, um die gewünschten ACID-Eigenschaften der globalen Transaktionen zu sichern.

Akzeptiert man andererseits eine Lockerung dieser Eigenschaften und erlaubt globale LSR-, 2LSR- oder QSR-Schedules, vereinfachen sich besonders die Aufgaben des GTM in der Föderation. Der in Abschnitt 5.2 erwähnte Top-Down-Ansatz zur Realisierung eines globalen 2LSR-Schedules überzeugt beispielsweise durch eine Vielzahl integrierbarer lokaler Systeme und das individuelle Ausnutzen lokaler Eigenschaften, ohne diese explizit zu fordern. Er garantiert somit bedeutend mehr lokale Autonomie als alle vorgestellten Ansätze zur Wahrung der globalen Serialisierbarkeit. Auch die Umsetzung von Recovery-Mechanismen beschränkt alternative Korrektheitsansätze nicht so stark wie die globalen Serialisierbarkeitsansätze.

Die mehrschichtigen Transaktionen bilden letztendlich einen Ansatz, der von einer völlige Lösung vom klassischen Transaktionsmodell ausgeht. Mit ihm ist es möglich, die Semantik einzelnen Operationen und Operationsfolgen auszunutzen, um so Alternativen zur klassischen Serialisierbarkeits- und Recovery-Theorie zu schaffen. Ihr Einsatz ist besonders mit Blick auf objektorientierte und lose gekoppelte Umgebungen interessant.

Kapitel 9

Zusammenfassung und Ausblick

Die vorliegende Arbeit hatte das Ziel, eine vergleichende Betrachtung aktueller Entwicklungen auf dem Gebiet der föderierten Transaktionsverwaltung durchzuführen. Dazu sollten bisherige Vorschläge zur Verwaltung von Transaktionen in föderierten Datenbankumgebungen analysiert, verglichen und bewertet werden und allgemeine Anforderungen an ein föderiertes Transaktionsmanagement für bestimmte Anwendungsszenarien abgeleitet werden. Die Probleme einer solchen Transaktionsverwaltung liegen in den drei geforderten Charakteristka föderierter Datenbanksysteme: der Heterogenität, der Verteilung und der Autonomie der an der Föderation beteiligten Datenbanksysteme. Unter Berücksichtigung dieser Eigenschaften soll ein FDBMS die volle Funktionalität klassischer DBMSe aufweisen, d.h. es sollen globale Transaktionen unterstützt werden, die den ACID-Eigenschaften genügen. aufweisen. Diese Forderung impliziert die Schaffung globaler Synchronisations- und Recovery-Mechanismen sowie Mechanismen zur Realisierung verklemmungsfreier globaler Abarbeitungen.

Bei der Analyse der existierenden Ansätze und Vorschläge wurden verschiedene Teilprobleme der föderierten Transaktionsverwaltung gesondert untersucht:

- Kriterien und Lösungsansätze zur Sicherung *global serialisierbarer* Abarbeitungen wurden vorgestellt und verglichen. Dabei wurde untersucht, inwieweit das Wissen über lokale Synchronisationsmechanismen genutzt werden kann, um das globale Transaktionsmanagement einfacher und effektiver zu gestalten.

- *Alternative Korrektheitskriterien* zur Synchronisation globaler Transaktionen wurden vorgestellt, die eine Abschwächung der Forderung nach globaler Serialisierbarkeit vornahmen, ohne in bestimmten Umgebungen die geforderte Korrektheit einer Abarbeitung zu beeinflußen.

- Verschiedene Ansätze zur Lösung der *Recovery-Problematik* wurden verglichen und ihre Kombinierbarkeit mit den Synchronisationsansätzen wurde untersucht.

- Möglichkeiten zur Realisierung einer globalen *Verklemmungsbehandlung* wurden vorgestellt.

Innerhalb dieser Arbeit erfolgte zunächst eine Einführung in den Bereich der FDBS, wobei die Terminologie und Methodik aus [SL90] verwendet wurde. Verschiedene weitere Arbeiten auf diesem Gebiet, z.B. [LMR90, HM85], wurden zur Betrachtung herangezogen und vergleichend ausgewertet.

Daraufhin erfolgte eine Einführung in die Grundlagen der Transaktionsverwaltung in föderierten Umgebungen. Zunächst wurden dazu allgemeine Grundbegriffe nach [BHG87] erläutert. Danach erfolgte die Darstellung eines allgemeinen Transaktionsmodells für FDBSe, das nachfolgend bei der Erläuterung der verschiedenen Ansätze als Referenzmodell genutzt wurde.

Bei der Untersuchung der Mechanismen zur Sicherung der Serialisierbarkeit globaler Abarbeitungen wurden zwei Hauptrichtungen betrachtet. Zum einen wurden die Möglichkeiten zur Integration von Systemen untersucht, die lediglich die Serialisierbarkeit und Rücksetzbarkeit der lokalen Abarbeitungen sichern. Mit den *Ticketmethoden* wurde ein entsprechender Ansatz dargestellt, der die Sicherung der gewünschten Eigenschaften bei einem Maximum an lokaler Autonomie realisierte. Andererseits wurde betrachtet, wie die Restriktion der CDBMSe auf Systeme mit bestimmten zusätzlichen lokalen Schedulereigenschaften die Aufgaben des globalen Transaktionsmanagements erleichtern kann. Die Forderung nach *Rigorosität* der lokalen Abarbeitungen stellte dabei eine sehr brauchbare und oft akzeptierte Restriktion der lokalen Autonomie dar. Aber auch die *strenge Rücksetzbarkeit*, die oft auch als *"Commitment Ordering"* bezeichnet wird, gilt in vielen Fällen als akzeptable Basis eines föderierten Transaktionsmanagements.

Die Restriktion der Eigenschaften lokaler Systeme bzw. die mangelhafte Leistungsfähigkeit föderierter Transaktionsverwaltungen mit serialisierbaren globalen Schedules war die Motivation zur Schaffung alternativer Korrektheitkriterien in föderierten Umgebungen. Innerhalb dieser Arbeit wurden mehrere alternative Ansätze und Kriterien vorgestellt und es wurde gezeigt, daß unter bestimmten Umständen die Lockerung der globalen Serialisierbarkeit durchaus sinnvoll sein kann. Mit der *zweistufigen Serialisierbarkeit* wurde dabei ein Kriterium vorgestellt, das unter Einhaltung bestimmter Zugriffsbeschränkungen im Hinblick auf die Umsetzung von Integritätsbedingungen die gleichen Eigenschaften realisierte, wie die globale Serialisierbarkeit. Die *lokale Serialisierbarkeit* und die *Quasiserialisierbarkeit* sicherten zwar nicht die Durchsetzung aller denkbaren Integritätsbedingungen, waren dafür aber in puncto Zugriffsbeschränkungen weit weniger restriktiv. Prinzipiell realisierten alle drei Kriterien eine größere Anzahl globaler Schedules als die globale Serialisierbarkeit. Ihr Einsatz ist besonders in lose gekoppelten Umgebungen denkbar, da in diesen eine Forcierung globaler Integritätsbedingungen nur in einem sehr geringem Umfang möglich ist.

Bei der Untersuchung der Recovery-Mechanismen wurde zunächst das *2PC-Protokoll* als gängigstes atomares Terminierungsprotokoll verteilter Datenbanksysteme vorgestellt. Da dieses Protokoll sehr restriktiv im Hinblick auf die integrierbaren Systeme ist, wurden alternative Ansätze zu diesem Protokoll untersucht. Dabei sind besonders die *2PCA-Methode* und der *"Commit"-Graphen-Ansatz* hervorzuheben, die jeweils vollständige Lösungen aller auftretenden Probleme lieferten. Es ist jedoch festzustellen, daß die Rea-

lisierung von Recovery-Mechanismen weitaus mehr Restriktionen bei der Autonomie der CDBSe verlangt als die Umsetzung von Synchronisationsmechanismen. Dies gilt insbesondere, wenn die Recovery-Ansätze mit Concurrency Control-Mechanismen zur Realisierung globaler Serialisierbarkeit kombiniert werden sollen. Der Einsatz von Kompensationstechniken verbunden mit einer Lockerung des Serialisierbarkeitsbegriffes erleichterte die Umsetzung von Recovery- und Synchronisationsmechanismen in föderierten Umgebungen. Alternative Transaktionsmodelle, wie das *mehrschichtige Transaktionsmodell*, realisierten so semantische Serialisier- und Rücksetzbarkeit. Ihr Einsatz scheint besonders in objektorientierten und lose gekoppelten Umgebungen sinnvoll.

Bei der Betrachtung globaler Verklemmungssituationen waren besonders die Wartebeziehungen interessant, die über die Grenzen der einzelnen Datenbankknoten hinausgingen. Aufgrund der lokalen Autonomie der CDBMSe stehen einer föderierten Transaktionsverwaltung lokale "wait-for"-Informationen nicht unbedingt zur Verfügung. Daraus ergab sich, daß die klassischen Methoden zur Verklemmungsbehandlung in verteilten Datenbanksystemen nicht für den Einsatz in föderierten Umgebungen geeignet waren. Mit dem *"Potential-Conflict"-Graphen-Ansatz* wurde eine Lösung vorgestellt, die Timeout-Mechanismen nutzte, um potentielle Wartebeziehungen in den einzelnen Knoten ausfindig zu machen.

Alle innerhalb dieser Arbeit vorgestellten Korrektheitskriterien, Mechanismen und Ansätze sind abschließend verglichen und mit Blick auf die Eignung zum Einsatz in föderierten Umgebungen bewertet worden. Eine klare Empfehlung eines föderierten Transaktionskonzepts konnte nicht gegeben werden. Die Gründe dafür liegen vor allem den in vielfältigen Einschränkungen, die die einzelnen Ansätze für die teilnehmenden Datenbanksysteme forderten. So wurde in vielen Fällen nicht nur die Design- sondern oftmals auch die Ausführungs- und Kommunikationsautonomie lokaler Datenbanksysteme beschränkt. Aus diesem Grund sind bei der Wahl eines Ansatzes für eine föderierte Transaktionsverwaltung das Einsatzszenario und vor allem die Präferenzen der Entwickler und Anwender von Bedeutung.

Zusammenfassend läßt sich feststellen, daß die Probleme der Umsetzung einer effizienten Transaktionsverwaltung in föderierten Umgebungen noch nicht befriedigend gelöst sind. Dies gilt vor allem in Hinblick auf die Realisierung von Mechanismen zur Gewährleistung der Atomarität und Fehlertoleranz globaler Transaktionen. Die zu dieser Problematik untersuchten Lösungen basierten alle auf sehr gravierenden Einschränkungen der Autonomie der lokalen Systeme. Besonders auf diesem Gebiet sind in Zukunft weiterführende Arbeiten nötig. Neben den in dieser Arbeit untersuchten föderierten Transaktionsverwaltungsmechanismen existiert eine ganze Reihe weiterer Lösungsvorschläge. Es wird demnach innerhalb dieser Arbeit nicht der Anspruch auf Vollständigkeit erhoben, jedoch sollten die wesentlichen Ideen und Richtungen bei der Realisierung eines föderierten Transaktionsmanagements in dieser Arbeit genannt worden sein. Die vorgestellten Ansätze sind alle theoretischer Natur. Eine Ausdehnung der Untersuchung auf bereits realisierte praktische Ansätze könnte zu klareren Vorstellungen von der Komplexität der Problematik und zu eindeutigeren Empfehlungen für den einen oder anderen

Ansatz führen.

Ein in [PBE95] durchgeführter Vergleich von elf objektorientierter MDBSen ergab beispielsweise, daß fünf der untersuchten Systeme gar kein Transaktionsmanagement unterstützten, während die anderen in der Regel erweiterte Transaktionsmodelle, z.B. das Flex-, ACTA- oder DOM-Transaktionsmodell[1], verwendeten, in denen der jeweilige Nutzer, ähnlich dem mehrschichtigen Transaktionsmodell, die Korrektheit einer Abarbeitung durch Beziehungen zwischen verschiedenen Subtransaktionen definieren konnte. Dies zeigt zum einen, daß sich die Konzepte für eine Transaktionsverwaltung in FDBSen noch in der Entwicklung befinden, andererseits aber ein Trend zur Lösung vom klassischen Transaktionsmodell zugunsten erweiterter Modelle erkennbar ist. Eine Betrachtung weiterer FDBS-Prototypen und der von ihnen realisierten Transaktionsmodelle könnte somit zu zusätzlichen Aspekten und Erkenntnissen führen.

[1] Diese und weitere erweiterte Transaktionsmodelle werden ausführlich in [Elm92] vorgestellt.

Literaturverzeichnis

[AGS87] R. Alonso, H. Garcia-Molina und K. Salem. Concurrency Control and Recovery for Global Procedures in Federated Database Systems. *Bulletin of the IEEE Technical Committee on Data Engineering*, 10(3):5–11, September 1987.

[AVA+94] G. Alonso, R. Vingralek, D. Agrawal, Y. Breitbart, A. El Abbadi, H.-J. Schek und G. Weikum. Unifying Concurrency Control and Recovery of Transactions. In M. Jarke und K. Jeffery, Herausgeber, *Advances in Database Technology — EDBT'94, Proc. of the 4th Int. Conf. on Extending Database Technology, Cambridge, UK*, S. 123–130. LNCS 779, Springer-Verlag, Berlin, März 1994.

[BG94] D. Barbará-Millá und H. Garcia-Molina. The Demarcation Protocol: A Technique for Maintaining Constraints in Distributed Database Systems. *The VLDB Journal*, 3(3):325–353, 1994.

[BGRS91] Y. Breitbart, D. Georgakopoulos, M. Rusinkiewicz und A. Silberschatz. On Rigorous Transaction Scheduling. *IEEE Transactions on Software Engineering*, 17(9):954–960, September 1991.

[BGS92] Y. Breitbart, H. Garcia-Molina und A. Silberschatz. Overview of Multidatabase Transaction Management. *The VLDB Journal*, 1(2):181–240, Oktober 1992.

[BHG87] P. A. Bernstein, V. Hadzilacos und N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, Reading, MA, 1987.

[BHP92] M. Bright, A. Hurson und S. Pakzad. A Taxonomy and Current Issues in Multidatabase Systems. *IEEE Computer*, 25(3):50–56, März 1992.

[BLS91] Y. Breitbart, W. Litwin und A. Silberschatz. Deadlock Problems in a Multidatabase Environment. In *36th IEEE Computer Society International Conference, San Francisco, Digest of Papers COMPCON*, S. 145–151. IEEE Computer Society Press, 1991.

[Bre90] Y. Breitbart. Multidatabase Interoperability. *ACM SIGMOD Record*, 19(3):53–60, 1990.

[BS88] Y. Breitbart und A. Silberschatz. Multidatabase Update Issues. In H. Boral und P.-A. Larson, Herausgeber, *Proc. of the 1988 ACM SIGMOD Int. Conf. on Management of Data, Chicago, IL, SIGMOD RECORD 17(3)*, S. 135–142. ACM Press, September 1988.

[BST90] Y. Breitbart, A. Silberschatz und G. R. Thompson. Reliable Transaction Management in a Multidatabase System. In H. Garcia-Molina und H. Jagadish, Herausgeber, *Proc. of the 1990 ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, SIGMOD RECORD 19(2)*, S. 215–224. ACM Press, Juni 1990.

[BST92] Y. Breitbart, A. Silberschatz und G. R. Thompson. Transaction Management Issues in a Failure-Prone Multidatabase System Environment. *The VLDB Journal*, 1(1):1–39, Juli 1992.

[DE89] W. Du und A. Elmagarmid. QSR: A Correctness Criterion For Global Concurrency Control in InterBase. In P. M. G. Apers und G. Wiederholt, Herausgeber, *Proc. of the 15th Int. Conf. on Very Large Data Bases (VLDB'89), Amsterdam, The Netherlands*, S. 347–355. Morgan Kaufmann Publishers, Palo Alto, CA, August 1989.

[DE90] W. Du und A. Elmagarmid. A Paradigm for Concurrency Control in Heterogeneous Distributed Database Systems. In M. Liu, Herausgeber, *Proc. of the 6th IEEE Int. Conf. on Data Engineering (ICDE'90), Los Angeles, CA, USA*, S. 37–46. IEEE Computer Society Press, Februar 1990.

[DSW94] A. Deacon, H.-J. Schek und G. Weikum. Semantics-based Multi-level Transaction Management in Federated Systems. In *Proc. of the 10th IEEE Int. Conf. on Data Engineering (ICDE'94), Houston, Texas, USA, 14–18 February 1994*, S. 452–461. IEEE Computer Society Press, Los Alamitos, CA, 1994.

[Elm92] A. K. Elmagarmid, Herausgeber. *Database Transaction Models For Advanced Applications.* Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[GRS91] D. Georgakopoulos, M. Rusinkiewicz und A. Sheth. On Serializability of Multidatabase Transactions through Forced Local Conflicts. In N. Cercone und M. Tsuchiya, Herausgeber, *Proc. of the 7th IEEE Int. Conf. on Data Engineering (ICDE'91), Kobe, Japan*, S. 314–323. IEEE Computer Society Press, April 1991.

[GRS94] D. Georgakopoulos, M. Rusinkiewicz und A. Sheth. Using Tickets to Enforce the Serializability of Multidatabase Transactions. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):119–153, April 1994.

[HM85] D. Heimbigner und D. McLeod. A Federated Architecture for Information Management. *ACM Transactions on Office Information Systems*, 3(3):253–278, Juli 1985.

[HS95] A. Heuer und G. Saake. *Datenbanken — Konzepte und Sprachen.* International Thomson Publishing, Bonn, 1995.

[Hsi92a] D. Hsiao. Federated Databases and Systems: Part I – A Tutorial on Their Data Sharing. *The VLDB Journal*, 1(1):127–179, Juli 1992.

[Hsi92b] D. Hsiao. Federated Databases and Systems: Part II – A Tutorial on Their Resource Consolidation. *The VLDB Journal*, 1(2):285–310, Oktober 1992.

[HTJ+95] M. Höding, C. Türker, S. Janssen, K.-U. Sattler, S. Conrad, G. Saake und I. Schmitt. Föderierte Datenbanksysteme — Grundlagen und Ziele des Projektes **SIGMA**$_{FDB}$. Preprint Nr. 12, Fakultät für Informatik, Universität Magdeburg, Dezember 1995.

[LKS91a] E. Levy, H. F. Korth und A. Silberschatz. A Theory Of Relaxed Atomicity. In *Proc. of the ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC), Montreal, Canada.* ACM Press, 1991.

[LKS91b] E. Levy, H. F. Korth und A. Silberschatz. An Optimistic Commit Protocol For Distributed Transaction Management. In J. Clifford und R. King, Herausgeber, *Proc. of the 1991 ACM SIGMOD Int. Conf. on Management of Data, Denver, Colorado, SIGMOD RECORD 20(2)*, S. 88–97. ACM Press, Juni 1991.

[LMR90] W. Litwin, L. Mark und N. Roussopoulos. Interoperability of Multiple Autonomous Databases. *ACM Computing Surveys*, 22(3):267–293, September 1990.

[MKSA92] J. G. Mullen, W. Kim und J. Sharif-Askary. On the Impossibility of Atomic Commitment in Multidatabase Systems. In P. A. Ng, C. V. Ramamoorthy, L. C. Seifert und R. T. Yeh, Herausgeber, *Proc. of The Second International Conference on System Integration (ICSI'92)*, S. 625–634. IEEE Computer Society Press, 1992.

[MR91] P. Muth und T. Rakow. Atomic Commitment For Integrated Database Systems. In N. Cercone und M. Tsuchiya, Herausgeber, *Proc. of the 7th IEEE Int. Conf. on Data Engineering (ICDE'91), Kobe, Japan*, S. 296–304. IEEE Computer Society Press, April 1991.

[MRB+92a] S. Mehrotra, R. Rastogi, Y. Breitbart, H. F. Korth und A. Silberschatz. Ensuring Transaction Atomicity In Multidatabase Systems. In *Proc. of the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of*

Database Systems (PODS), San Diego, CA, S. 164–175. ACM Press, Juni 1992.

[MRB+92b] S. Mehrotra, R. Rastogi, Y. Breitbart, H. F. Korth und A. Silberschatz. The Concurrency Problem in Multidatabases: Characteristics and Solutions. In M. Stonebraker, Herausgeber, *Proc. of the 1992 ACM SIGMOD Int. Conf. on Management of Data, San Diego, CA, SIGMOD RECORD 21(2)*, S. 288–297. ACM Press, Juni 1992.

[MVN93] P. Muth, J. Veijalainen und E. Neuhold. Extending Multi-Level Transactions For Heterogeneous And Autonomous Database Systems. GMD Technical Report (Arbeitspapiere der GMD) No. 739, Sankt Augustin, GMD Institute for Integrated Publication and Information Systems (IPSI), Darmstadt, März 1993.

[OAB94] M. Ouzzani, M. A. Atroun und N. L. Belkhodia. A Top-Down Approach For Two Level Serializability. In J. B. Bocca, Matthias Jarke und C. Zaniolo, Herausgeber, *Proc. of the 20th Int. Conf. on Very Large Data Bases (VLDB'94), Santiago, Chile*, S. 226–237. Morgan Kaufmann Publishers, San Francisco, CA, September 1994.

[ÖV91] M. T. Özsu und P. Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, Eaglewood Cliffs, New Jersey, 1991.

[PBE95] E. Pitoura, O. Bukhres und A. K. Elmagarmid. Object Orientation in Multidatabase Systems. *ACM Computing Surveys*, 27(2):141–195, Juni 1995.

[PL91] C. Pu und A. Leff. Replica Control in Distributed Systems: An Asynchronous Approach. In J. Clifford und R. King, Herausgeber, *Proc. of the 1991 ACM SIGMOD Int. Conf. on Management of Data, Denver, Colorado, SIGMOD RECORD 20(2)*, S. 377–386. ACM Press, Juni 1991.

[Pu88] C. Pu. Superdatabases for Composition of Heterogeneous Databases. In J. Carlis, Herausgeber, *Proc. of the 4th IEEE Int. Conf. on Data Engineering (ICDE'88), Los Angeles, CA, USA*, S. 548–555. IEEE Computer Society Press, Februar 1988.

[Rah94] E. Rahm. *Mehrrechner-Datenbanksysteme.* Addison-Wesley, Bonn, 1994.

[Raz92] Y. Raz. The Principle of Commitment Ordering, or Guaranteeing Serializability in a Heterogeneous Environment of Multiple Autonomous Resource Managers Using Atomic Commitment. In L.-Y. Yuan, Herausgeber, *Proc. of the 18th Int. Conf. on Very Large Data Bases (VLDB'92), Vancouver, Canada*, S. 292–312. Morgan Kaufmann Publishers, San Mateo, CA, August 1992.

[SGA89] K. Salem, H. Garcia-Molina und R. Alonso. Altruistic Locking: A Strategy For Coping Long Lived Transactions. In K. R. Dittrich, U. Dayal und A. P. Buchmann, Herausgeber, *High Performance Transaction Systems*, S. 176–199. LNCS 359, Springer-Verlag, New York, 1989.

[SL90] A. P. Sheth und J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[SS93] W. Schaad und H.-J. Schek. Federated Transaction Management Using Open Nested Transactions. In *Proc. of the DBTA Workshop on Interoperability of Database Systems and Database Applications, Fribourg, Switzerland*, Oktober 1993.

[SSW95] W. Schaad, H.-J. Schek und G. Weikum. Implementation and Performance of Multi-level Transaction Management in a Multidatabase Environment. In O. Bukhres, T. Özsu und M.-C. Shan, Herausgeber, *Proc. of the 5th Int. Workshop on Research Issues on Data Engineering: Distributed Object Management (RIDE-DOM'95), Taipei, Taiwan*, S. 108–115. IEEE Computer Society Press, Los Alamitos, CA, März 1995.

[SWS91] H.-J. Schek, G. Weikum und W. Schaad. A Multi-level Transaction Approach to Federated DBMS Transaction Management. In Y. Kambayashi, M. Rusinkiewicz und A. Sheth, Herausgeber, *Proc. of the 1st Int. Workshop on Interoperability in Multidatabase Systems (IMS'91), Kyoto, Japan*, S. 280–287. IEEE Computer Society Press, 1991.

[SWY93] H.-J. Schek, G. Weikum und H. Ye. Towards a Unified Theory of Concurrency Control and Recovery. In *Proc. of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), Washington, DC*, S. 300–311. ACM Press, Mai 1993.

[VG93] G. Vossen und M. Groß-Hardt. *Grundlagen der Transaktionsverarbeitung.* Addison-Wesley, Bonn, 1993.

[Vos94] G. Vossen. *Datenbank-Sprachen und Datenbank-Management-Systeme.* Addison-Wesley, Bonn, 2 Auflage, 1994.

[VW92] J. Veijalainen und A. Wolski. Prepare and Commit Certification for Decentralized Transaction Management in Rigorous Heterogeneous Multidatabases. In F Golshani, Herausgeber, *Proc. of the 8th IEEE Int. Conf. on Data Engineering (ICDE'92), Tempe, AR, USA*, S. 470–480. IEEE Computer Society Press, Los Alamitos, CA, Februar 1992.

[WDSS93] G. Weikum, A. Deacon, W. Schaad und H. Schek. Open Nested Transactions in Federated Database Systems. *Bulletin of the IEEE Technical Committee on Data Engineering*, 16(2):4–7, Juni 1993.

[WS92] G. Weikum und H.-J. Schek. Concepts and Applications of Multi-level Transactions and Open Nested Transactions. In A. K. Elmagarmid, Herausgeber, *Database Transaction Models for Advanced Applications*, S. 515–553. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[WV90] A. Wolski und J. Veijalainen. 2PC Agent Method: Achieving Serializability in Presence of Failures in a Heterogeneous Multidatabase. In *Proc. of IEEE PARBASE-90 Conf., Miami Beach, Florida*, S. 321–330, März 1990.

[ZE93] A. Zhang und A. K. Elmagarmid. A Theory of Global Concurrency Control in Multidatabase Systems. *The VLDB Journal*, 2(3):331–360, Juli 1993.

Selbständigkeitserklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbständig und nur mit erlaubten Hilfsmitteln angefertigt habe.

Magdeburg, den 29. August 1996

Thomas Lehnecke

Thesen

1. Eine Transaktionsverwaltung im klassischen Sinn, d.h. die Realisierung der ACID-Eigenschaften von lokalen und globalen Transaktionen, ist in föderierten Datenbanksystemen ohne Einschränkungen der Autonomie der lokalen Datenbanksysteme nicht möglich.

2. Die Forcierung der CCSR-, SSR- oder HSR-Eigenschaft globaler Schedules kann die globale Serialisierbarkeit sichern, ohne die lokale Autonomie einzuschränken. Folglich gilt es, einen Synchronisationsmechanismus zu schaffen, der in fehler- und abbruchfreien Umgebungen die Isolations- und Konsistenz-Eigenschaften globaler Transaktionen realisiert und trotzdem die lokale Autonomie wahrt.

3. Die Voraussetzung zusätzlicher lokaler Scheduleeigenschaften zu Synchronisationszwecken ist in vielen Anwendungsszenarien eine akzeptable Einschränkung der lokalen Autonomie. Dies gilt vor allem, wenn lediglich klassische kommerzielle Datenbanksysteme föderiert werden sollen.

4. Die Rigorosität lokaler Abarbeitungen wird von einer Vielzahl klassischer Datenbanksysteme garantiert und bildet somit eine gute Grundlage, um einen einfachen globalen Synchronisationsmechanismus zu realisieren.

5. Alternative Korrektheitkriterien sind in einer Reihe von Anwendungsszenarien, besonders in lose gekoppelten FDBSen, akzeptabel und können den Parallelitätsgrad und den Transaktionsdurchsatz des föderierten Datenbanksystems erhöhen.

6. Die Maßnahmen zur Realisierung von Recovery-Mechanismen sind viel restriktiver als die Maßnahmen zur Durchsetzung globaler Serialisierbarkeit und verletzen die geforderten charakteristischen Eigenschaften föderierter Systeme.

7. Die Anwendung eines 2PC-Protokolls in FDBSen widerspricht der Forderung nach lokaler Autonomie und damit der grundlegenden Charakteristik eines FDBSs.

8. Alternativen zum 2PC-Protokoll weichen entweder die Atomarität und Dauerhaftigkeit einer globalen Transaktion auf oder fordern gravierende Einschränkungen bei der Design- und Ausführungsautonomie der lokaler Systeme.

9. Erweiterte Transaktionsmodelle, die semantikbasierte Synchronisations- und Recovery-Mechanismen nutzen, spielen für die föderierte Transaktionsverwaltung zunehmend eine größere Rolle. Ihre Anwendung ist besonders in lose gekoppelten und objektorientierten Umgebungen ratsam.

10. Durch die Forderung nach lokaler Design- und Ausführungsautonomie und die damit verbundene Abwesenheit von Informationen über lokale Wartebeziehungen können globale Verklemmungen nur näherungsweise bestimmt (geschätzt) werden.

11. Das Gebiet der föderierten Datenbanksysteme ist ein aktueller Arbeitsbereich der Datenbankforschung. Ein optimales föderiertes Transaktionsmanagement, das alle geforderten Eigenschaften aufweist, wurde noch nicht gefunden. Weitere Realisierungsansätze für eine föderierte Transaktionsverwaltung werden durch die Entwicklung von Forschungsprototypen und kommerziellen Systemen auftauchen und für ausführliche Untersuchungen zur Verfügung stehen.