OTTO-VON-GUERICKE-UNIVERSITÄT MAGDEBURG

Fakultät für Informatik
Institut für Technische Informationssysteme

Diplomarbeit

Grundlagen der Konsistenzsicherung in heterogenen, föderierten Datenbankumgebungen

Verfasser:
Guido Grohmann

Betreuer:
Prof. Dr. habil. G. Saake
Dipl.-Inform. C. Türker

Grohmann, Guido:

Grundlagen der Konsistenzsicherung in heterogenen, föderierten Datenbankumgebungen: Diplomarbeit. - Otto v. Guericke-Universität Magdeburg, 1995.

Erklärung

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

Magdeburg, den 18.09.1995 (Guido Grohmann)

Vorwort

Die vorliegende Arbeit wurde im Institut für Technische Informationssysteme der Fakultät für Informatik der Otto v. Guericke-Universität Magdeburg im Sommersemester 1995 angefertigt. Ziel der Arbeit war es, die Grundlagen aktiver Konsistenzsicherung in heterogenen, föderierten Datenbankumgebungen zu erarbeiten. Hierzu war zunächst eine Einarbeitung in die Themenbereiche Konsistenzsicherung in Datenbanken und föderierte Datenbanksysteme nötig. In dieser Arbeit sollte untersucht werden, wie aktive Mechanismen zur Konsistenzsicherung in föderierten Datenbankumgebungen eingesetzt werden können. Ein weiterer Schwerpunkt der Arbeit bildet die Untersuchung der Abbildung aktiver, datenbankübergreifender Mechanismen auf vorhandene Konsistenzsicherungsmechanismen in einem föderierten Datenbanksystem. Hierzu sollte insbesondere diese Abbildung für ein relationales sowie ein objektorientiertes Datenbanksystem betrachtet werden.

Danksagung

Hiermit möchte ich mich bei meinem Betreuer Dipl.-Inform. Can Türker für die geduldige und hilfreiche Unterstützung bedanken. Meinem Kommilitonen Eyk Hildebrandt danke ich für die unterstützenden Diskussionen. Abschließend sei auch Herrn Prof. Dr. habil. Gunter Saake gedankt.

Inhaltsverzeichnis

1 Einleitung 7

2 Föderierung von heterogenen, autonomen DBSen 9

2.1 Charakterisierung von föderierten Datenbanksystemen 10

2.2 Zwei unterschiedliche Architekturkonzepte von föderierten Datenbanksystemen 14

2.3 Beispielszenarien für Föderationen von Datenbanksystemen 16

2.3.1 Beispielszenarien für eine lose gekoppelte Föderation 16

2.3.2 Beispielszenario für eine eng gekoppelte Föderation 18

3 Grundlagen der Konsistenzsicherung in DBSen 24

3.1 Konsistenzbedingungen in zentralen Systemen 24

3.1.1 Konsistenzbedingungen in relationalen Datenbanksystemen 25

3.1.2 Konsistenzbedingungen in objektorientierten Datenbankmanagementsystemen 30

3.2 Das ECA-Regel Konzept 32

3.3 Konsistenzbedingungen in föderierten Systemen 36

4 Mechanismen zur Konsistenzsicherung in FDBSen 39

4.1 Realisierung global definierter Konsistenzbedingungen mittels lokal vorhandener Konsistenzsicherungsmechanismen 40

4.1.1 Domain- bzw. Attributbedingungen 41

4.1.2 Konsistenzbedingungen auf Objektebene 41

4.1.3 Globale Schlüsselbedingung 43

4.1.4 Globale Aggregatbedingungen 45

4.1.5 Rekursive Bedingungen 46

4.1.6 Referentielle Bedingungen 46

4.2 ECA-Regeln in föderierten Systemen ... 48

4.2.1 Ereignisse in föderierten Umgebungen ... 49

4.2.2 Systemübergreifende Bedingungen von ECA-Regeln ... 50

4.2.3 Systemübergreifende Aktionen ... 50

4.3 Architektur eines Regelverarbeitungssystems in FBDSen ... 51

4.3.1 Architekturvorschlag ... 51

4.3.2 Ereignisdetektion in föderierten Systemen ... 53

4.4 Ableitung lokaler Regeln aus global definierten Regeln ... 54

4.4.1 Realisierungsvorschlag für die Ableitung ... 55

4.4.2 Trigger als lokale Regeln ... 58

4.5 Verbesserung des Ableitungsprozesses ... 59

4.5.1 Optimierung der Ereignisdetektion ... 59

4.5.2 Weitergehende Optimierungsversuche ... 62

5 Probleme der Konsistenzsicherung in föderierten DB ... 66

5.1 Konsistenzbegriff in föderierten Datenbanksystemen ... 66

5.2 Probleme bei der Konsistenzsicherung durch Autonomieforderungen ... 68

5.3 Ereigniskomposition in föderierten Datenbanksystemen unter dem Einfluß der Kommunikationsdauer ... 70

5.4 Zeitabhängige Gültigkeitsbetrachtung von Konsistenzbedingungen ... 71

6 Schlußbetrachtung und Ausblick ... 74

Thesen ... 76

Quellenverzeichnis ... 77

1 Einleitung

Die Entwicklung der Datenbanktechnologie hat eine Vielzahl verschiedener Datenbanksysteme, die auf verschiedenen Datenmodellen basieren, hervorgebracht. Es ist daher keine Seltenheit, daß größere Unternehmen in ihren einzelnen Abteilungen unterschiedliche Datenbanksysteme einsetzen, die unabhängig voneinander eingeführt und weiterentwickelt wurden. Aus dieser Situation ergeben sich einige Probleme, wie z.B. redundante Datenhaltung oder Datenaustausch zwischen den möglicherweise völlig verschiedenen Datenbanksystemen. Zu diesen vorhandenen Datenbanksystemen gibt es eine Vielzahl von bereits bestehenden Anwendungen, die wie gewohnt weiterhin genutzt werden sollen. Darüber hinaus ist jedoch für neu zu erstellende Anwendungsprogramme ein transparenter Zugriff über eine einheitliche Schnittstelle auf alle heterogenen Daten wünschenswert.

Das Konzept der föderierten Datenbanken soll o.g. Probleme lösen. Ein solches föderiertes Datenbanksystem besteht aus möglicherweise heterogenen, autonomen Datenbanksystemen [SL90]. Auf diesen lokalen Datenbanksystemen können bereits bestehende, lokale Anwendungen laufen. Ein föderiertes Datenbanksystem erlaubt u.a. die Ausführung systemübergreifender globaler Anwendungen. Im Zuge der Forschungsarbeit werden verschiedene Projekte wie TSIMMIS [GPQ+94], EFENDI [Rad94, BEK94] und IRO-DB [BFHK94, GGF94] bearbeitet, in deren Verlauf bereits Forschungsprototypen mit teilweise eingeschränkter Funktionalität entstanden sind.

In föderierten Datenbanksystemen besteht (wie in zentralisierten Datenbanksystemen) die Forderung nach Konsistenzsicherung für die föderierten Daten. Während in zentralisierten Datenbanksystemen das zugehörige Datenbankmanagementsystem die Konsistenz garantieren kann, müssen föderierte Datenbanksysteme Mechanismen zur Erhaltung der globalen Konsistenz bereitstellen. Zur Überwachung solcher globaler Konsistenzbedingungen sollen aktive Mechanismen eingesetzt werden, wie sie bisher nur in aktiven zentralisierten Datenbanksystemen zur Konsistenzsicherung benutzt werden [GDD94, ZDDM94].

In dieser Arbeit werden die Grundlagen der Konsistenzsicherung in föderierten Datenbanksystemen erarbeitet. Hierzu wird angestrebt, globale Konsistenzbedingungen zu überwachen. Es wird dazu untersucht, inwieweit solche Konsistenzbedingungen sich auf Daten in verschiedenen Datenbanksystemen beziehen können. Die Realisierung der Überwachung solcher systemübergreifender Konsistenzbedingungen soll unter Benutzung der Konsistenzsicherungmechanismen der vorhandenen Datenbanksysteme erfolgen. Die Nutzung vorhandener Mechanismen wird angestrebt, um im Unterschied zur Einführung neuer Mechanismen, eine höhere Effizienz zu erzielen. Außerdem kommt dieser Sachverhalt den Autonomieforderungen der am föderierten Datenbanksystem beteiligten Datenbanksysteme

entgegen, da die zu nutzenden vorhandenen Konsistenzmechanismen auf jeweils eines der vorhandenen Datenbanksysteme beschränkt sind.

In den nachfolgenden Kapiteln 2 und 3 werden die Grundlagen der föderierten Datenbanksysteme sowie der Konsistenzsicherung in Datenbanksystemen allgemein beschrieben. Insbesondere werden im Kapitel 3 Konsistenzbedingungen für föderierte Systeme betrachtet. Im Kapitel 4 wird dann das ECA-Regel[1] Konzept im Kontext föderierter Systeme besprochen. Ferner wird ein Architekturansatz für ein (aktives) Konsistenzsicherungssystem in föderierten Umgebungen beschrieben. Globale Konsistenzbedingungen werden unter Zuhilfenahme von ECA-Regeln formuliert. Diese Regeln müssen dann auf lokale Konsistenzsicherungsmechanismen abgebildet werden. Dazu wird anschließend ein informeller Algorithmus zur Umsetzung von ECA-Regeln in eine zur Verarbeitung durch lokale Konsistenzsicherungsmechanismen geeignete Form formuliert. Außerdem erfolgt die Betrachtung von Optimierungsmöglichkeiten für diesen Algorithmus.

Im Kapitel 5 werden dann aus dem in dieser Arbeit gewählten Ansatz zur Konsistenzsicherung resultierende Probleme der Konsistenzsicherung in föderierten Datenbanksystemen diskutiert. Dazu gehören Zeitprobleme sowie auch Probleme mit Autonomieforderungen der an der Föderation beteiligten Datenbanksysteme. Ein Ausblick schließt die Arbeit ab.

[1]ECA-Regeln [DBB+88]: **E**VENT-**C**ONDITION-**A**CTION Regeln - ein universelles Konzept zur Realisierung aktiver Funktionalität von Datenbanksystemen

2 Föderierung von heterogenen, autonomen DBSen

Die Entwicklung der Datenbanktechnologie hat unterschiedliche Datenbanksysteme für die verschiedensten Anwendungszwecke hervorgebracht. In vielen Unternehmungen wurde daraufhin versucht, dem technischen Stand gemäß zu jedem Zeitpunkt einer Systemeinführung die jeweils für den speziellen Anwendungsfall besonders geeignete Datenbanktechnologie einzusetzen. Ein daraus entstehender Nebeneffekt ist die Existenz oftmals heterogener Datenbanksysteme mit redundanten Daten innerhalb eines Unternehmens.

Aus diesem Zustand resultieren die Bemühungen, heterogene Informationssysteme zu einem umfassenden System zu föderieren, auf welches über eine einheitliche Schnittstelle zugegriffen werden kann. Bei der Einrichtung eines föderierten Systems ist zu beachten, daß bereits bestehende Datenbankanwendungen davon unberührt weiterhin genutzt werden können. Dies ist besonders dann wichtig, wenn das vorhandene Datenbanksystem maßgeschneiderte Technologien bereitstellt, die das Anwendungsprogramm benutzt. Eine mögliche Alternative zur Schaffung einer einheitlichen Schnittstelle zum Zugriff auf alle Daten eines Unternehmens ist die Schaffung eines homogenen Systems, in das sowohl die Daten als auch die Anwendungsprogramme migrieren sollen. Dies bedeutet, daß bereits bestehende Anwendungsprogramme auf das neu geschaffene System portiert werden sollen. Hierbei könnte z.B. der Fall eintreten, daß sich aufgrund fehlender Quelltexte und damit ohne eine Möglichkeit einer Neuübersetzung dieser Anwendungen eine Portierung verbietet. Eine Portierung bestehender Anwendungsprogramme kann auch unökonomisch sein, wenn z.B. in absehbarer Zeit diese Anwendungen überflüssig sind oder eine Neuprogrammierung vorgesehen ist. Aus diesen Gründen ist die Schaffung eines föderierten Systems mit autonomen Teilsystemen günstiger, da bei diesem Ansatz die bereits auf den vorhandenen Datenbanksystemen laufenden Anwendungsprogramme weiterhin arbeiten können.

Ein *föderiertes Datenbanksystem* (FDBS) ist eine Sammlung von kooperierenden, autonomen und möglicherweise auch heterogenen Datenbanksysteme (vgl. dazu [SL90]). Diese einzelnen Datenbanksysteme werden als Komponentendatenbanksysteme (CDBS) bezeichnet. Ein Komponentendatenbanksystem kann mit verschiedenen föderierten Datenbanksystemen zusammenarbeiten. Die Föderation als Kooperation zwischen unabhängigen Systemen bedeutet eine gesteuerte und manchmal auch limitierte Integration von Komponentendatenbanksystemen in einem oder mehreren föderierten Datenbanksystemen.

In diesem Kapitel sollen zunächst die für die weitere Arbeit wichtigen Eigenschaften der föderierte Datenbanksysteme erklärt werden. Im Abschnitt 2.2 werden zwei wichtige Architekturen für föderierte Datenbanksysteme vorgestellt. Daraufhin sollen dann Beispielszenarien vorgestellt werden, die die Anwendung der beiden Architekturkonzepte

beschreiben. Gleichzeitig dienen diese Szenarien zur Aufstellung von Konsistenzbedingungen, für deren Wartung zu einem späteren Zeitpunkt in dieser Arbeit Konzepte und Mechanismen vorgestellt werden sollen.

2.1 Charakterisierung von föderierten Datenbanksystemen

Föderierte Datenbanksysteme können durch drei orthogonale, d.h. voneinander unabhängige Dimensionen oder Eigenschaften charakterisiert werden. Diese Eigenschaften sind:

- Datenverteilung,
- Heterogenität und
- Autonomie der Komponentendatenbanksysteme.

Bei der Föderierung bestehender Systeme kann es auftreten, daß bereits in diesen Systemen enthaltene Daten über mehrere Datenbanken verteilt vorliegen. Innerhalb des gesamten föderierten Systems entsteht die Datenverteilung durch die Existenz mehrerer Komponentendatenbanksysteme. Diese Datenverteilung muß bei einer globalen Konsistenzsicherung bekannt sein, um die Daten dann auch konsistent halten zu können. Wenn nicht bekannt ist, wo die zu überwachenden Daten denn nun eigentlich abgelegt sind, ist ihre Konsistenzsicherung auch nicht möglich. Diese Informationen werden bei der Kopplung (Integration) der Komponentensysteme als Metadaten im Data-Dictionary des föderierten Systems abgelegt.

Die im Verlauf der Entwicklung der Datenbanktechnologie entstandenen Datenbanksysteme weisen viele unterschiedliche Merkmale auf. Dadurch gibt es einige verschiedene Arten von Heterogenität zwischen den existierenden Systemen, z.B. in Bezug auf das zugrundeliegende Datenmodell, die Anfragesprache oder die Transaktionsverwaltung.

Die Autonomie von Teilsystemen ergibt sich zum einen aus der schon erwähnten Notwendigkeit der Weiterverwendung bestehender Anwendungsprogramme, die aufgrund fehlender Quelltexte oder absehbarer Neuprogrammierung nicht portiert werden können oder sollen. Desweiteren gibt es aber auch föderierte Systeme, die nur lesenden Datenzugriff auf föderierter Ebene bieten wollen. Hier sind die Komponentensysteme autonom, um die wichtigen Daten in den Komponentensystemen vor unbeabsichtigter oder unbefugter Veränderung durch auf föderierter Ebene laufende Anwendungen zu schützen. Datenmanipulation können dann nur lokal erfolgen.

Heterogenität der Komponentendatenbanksysteme

Die Heterogenität entsteht zum einen durch Verwendung unterschiedlicher Datenbankmanagementsysteme in den Komponentendatenbanksystemen, zum anderen durch die verschiedene Darstellung semantisch gleichwertiger Daten. Letztere Form der Heterogenität wird auch als semantische Heterogenität bezeichnet.

Heterogenität zwischen Datenbanksystemen aufgrund unterschiedlicher Datenbankmanagementsysteme entsteht durch Unterschiede z.B. zwischen Datenmodellen und Datenstrukturen, Konsistenzsicherungsmöglichkeiten oder Anfragesprachen. In bestehenden Datenbanksystemen gibt es unterschiedliche Strukturen und Datenmodelle. So wurden das Netzwerkmodell, das relationale und objektorientierte Modelle entwickelt. In Datenbanksystemen, die auf verschiedenen Datenmodellen basieren, gibt es auch verschiedene, oftmals an das Datenmodell gebundene Konsistenzsicherungsmechanismen.

Konsistenzsicherung wird deshalb in jedem Datenbanksystem unterschiedlich unterstützt. Konsistenzbedingungen in dem einen Datenmodell müssen in einem anderen Modell durch Verwendung anderer und eventuell mehrerer Mechanismen gemeinsam nachgebildet werden. Bei der Schaffung von globalen Konsistenzsicherungsmechanismen in einem föderierten Datenbanksystem unter Verwendung bereits in Komponentendatenbanksystemen bestehender Mechanismen ist diese Form der Heterogenität zu beachten. So kann z.B. die Realisierung einer dem CODASYL-Settyp vergleichbaren Bedingung in einem relationalen Datenbankmanagementsystem durch Verwendung der referentiellen Integrität und zusätzliche Trigger erfolgen.

Der Zugriff auf die Daten eines Datenbanksystems erfolgt über eine Anfragesprache, wie z.B. SQL. Es gibt kommerzielle Datenbankmanagementsysteme, die auf demselben Datenmodell basieren, und mit unterschiedlichen Anfragesprachen ausgestaltet sind. Oftmals gibt es auch Unterschiede durch herstellereigene Erweiterungen. So sind auf verschiedenen relationalen Datenbankmanagementsystem unterschiedliche Versionen der Sprache SQL mit jeweils eigenen Erweiterungen oder auch die Sprache QUEL anzutreffen.

Semantische Heterogenität hingegen entsteht durch Mißverständnisse beim Gebrauch, hinsichtlich der Bedeutung oder unterschiedlicher Interpretationen derselben oder gleichbedeutender Daten. Diese Form der Heterogenität ist auch zwischen Schemata ein- und desselben föderierten Datenbanksystems zu finden, wenn dieses mehrere Schemata besitzt.

Beispiel 2.1: In zwei Verkaufsdatenbanken DB_Shop1 und DB_Shop2 seien Waren mit den dazugehörenden, in beiden Geschäften jedoch teilweise unterschiedlichen Preisen enthalten. In der Datenbank DB_Shop1 sei die Mehrwertsteuer im Preis enthalten, während in DB_Shop2 alle Preise ohne Mehrwertsteuer angegeben sind. Ein direkter Vergleich

dieser gespeicherten Preise für ein- und dasselbe Produktes in beiden Datenbanken ergebe ein falsches, verzerrtes Ergebnis.

Autonomieforderungen von Komponentendatenbanksystemen

Die Autonomie der Komponentendatenbanksysteme kann unter verschiedenen Gesichtspunkten betrachtet werden. Aus diesem Grund gibt es folgende Formen der Autonomie: Designautonomie, Kommunikationsautonomie, Ausführungsautonomie und Assoziationsautonomie (vgl. [ÖV91]).

Zunächst wird die *Designautonomie* betrachtet. Für die einzelnen Komponentendatenbanken ist es natürlich, daß für sie unabhängig voneinander Designentscheidungen getroffen wurden bzw. werden. Dies resultiert aus der Tatsache, daß vor dem Aufbau eines föderierten Datenbanksystems mehrere voneinander völlig unabhängige Datenbanksysteme existieren. Designentscheidungen können die folgenden Sachverhalte betreffen:

- den Diskursbereich der Datenbank,
- die Repräsentation und die Benennung von Datenelementen (abhängig vom gewählten Datenmodell und der zu nutzenden Anfragesprache),
- die semantische Interpretation der Daten und
- die Art und der Umfang der Verwendung vorhandener Konsistenzsicherungsmöglichkeiten.

Eine weitere Möglichkeit der Forderung nach Autonomie ist die *Kommunikationsautonomie*. Die Kommunikationsautonomie ist die Fähigkeit, über den Zeitpunkt und die Art der Reaktion auf Anfragen anderer Datenbankmanagementsysteme autonom zu entscheiden.

Der *Ausführungsautonomie* liegt die Forderung zugrunde, daß die Ausführung lokaler Operationen (von lokalen, d.h. nur auf dem Komponentendatenbanksystem laufenden Anwendungen) auf einem Komponentendatenbanksystem ohne Wechselwirkungen mit externen Datenbankoperationen (z.B. vom föderierten Datenbanksystem) stattfinden. Konsistenzmechanismen können mit dieser Form der Autonomie in Konflikt kommen, wenn zur Überprüfung global definierter Konsistenzbedingungen lokale Daten gelesen oder modifiziert werden sollen. Das Datenbankmanagementsystem entscheidet dabei auch über die Reihenfolge der Bearbeitung mehrerer anstehender externer Anfragen. Sollten bei der Bearbeitung externer Operationen lokale Konsistenzbedingungen verletzt werden, kann das

lokale Datenbankmanagementsystem bei totaler Ausführungsautonomie diese Operation ohne Nachricht an das föderierte Datenbanksystem abbrechen. Lokale Operationen sollen also unbeeinflußt von der Zugehörigkeit des Datenbanksystems zu einem föderierten Datenbanksystem durchgeführt werden. Externe und lokale Operationen werden vom Komponentensystem gleichberechtigt behandelt.

Die *Assoziationsautonomie* ist eine weitere mögliche Form der Autonomie. Diese Form der Autonomie bezieht sich auf die Entscheidung, welche Funktionen und Ressourcen Fremdnutzern zur Verfügung gestellt werden sollen. Dies schließt auch Beschränkungen oder strikte Verbote solcher Zugriffe ein. Oftmals wird die Assoziationsautonomie auch als Teil der Designautonomie angesehen.

2.2 Zwei unterschiedliche Architekturkonzepte von föderierten Datenbanksystemen

Im vorangegangenen Abschnitt wurden föderierte Datenbanksysteme allgemein charakterisiert. In diesem Abschnitt werden nun zwei Architekturvorschläge für föderierte Datenbanksysteme diskutiert.

In [SL90] werden föderierte Datenbanksysteme in zwei Architekturkonzepte klassifiziert: *lose gekoppelte* föderierte Datenbanksystem und *eng gekoppelte* föderierte Datenbanksystem. Diese Konzepte basieren auf der Verantwortlichkeit für die Föderation und die Art der Integration der Komponentensysteme.

In lose gekoppelten föderierten Datenbanksystemen ist der Anwendungsentwickler für die Schaffung und Wartung der Föderation zuständig. In eng gekoppelten föderierten Datenbanksystemen hingegen haben die globalen und lokalen Datenbankadministratoren die Kontrolle über die Schaffung und den Betrieb des föderiertes Systems.

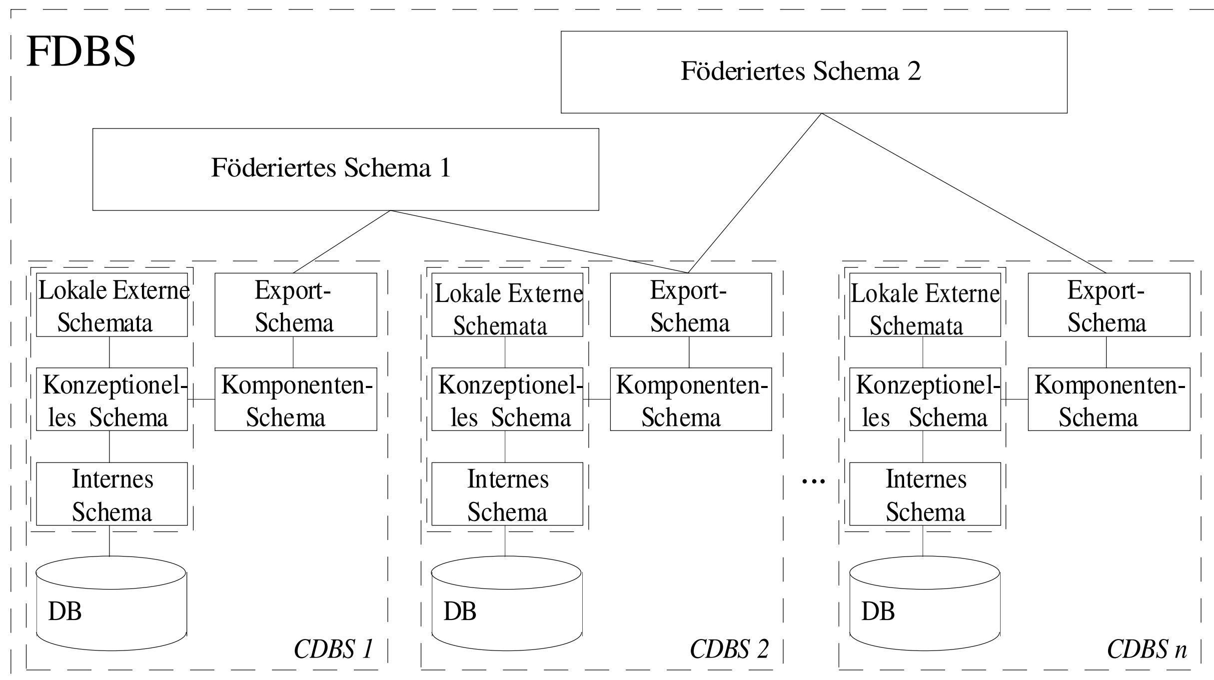

Abb. 2.1 FDBS mit mehreren föderierten Schemata - lose gekoppelter Ansatz

In beiden föderierten Systemen wird aufgrund der Assoziationsautonomie über die Zugänglichkeit von Daten bzw. Teilen oder die Bereitstellung von Funktionen des Komponentendatenbanksystem durch die lokalen Administratoren dieser Systeme entschieden. Die Erstellung einer Föderation erfolgt durch selektive und kontrollierte Integration ihrer Komponenten. Für jedes der Komponentensysteme werden Exportschemata erstellt, die dann zu einem oder auch mehreren föderierten Schemata integriert werden können. Die

Exportschemata entstehen durch einen Filterprozess auf den zugehörigen Komponentenschemata. Für jedes Komponentenschema können mehrere Exportschemata entstehen, wenn mehrere föderierte Schemata aufgebaut werden sollen. Die Erstellung dieser Komponentenschemata erfolgt durch Translation der konzeptionellen Schemata der Komponentendatenbanksysteme in die Datenmodellsprache des föderierten Schemas.

Für die Translation und die Filterung sind die jeweiligen Datenbankadministratoren der lokalen Datenbanksysteme verantwortlich. In der Art und Weise der Durchführung der Integration der Exportschemata in ein oder mehrere föderierte Schemata unterscheiden sich die beiden Architekturkonzepte. Bei einem eng gekoppelten Architekturansatz erfolgt die Integration der Exportschemata in ein oder mehrere föderierte Schemata durch einen Abstimmungsprozess zwischen den lokalen Administratoren und dem globalen Administrator. In einem lose gekoppelten föderierten Datenbanksystem können aus der Menge der durch die lokalen Administratoren angebotenen Exportschemata der Komponentendatenbanksysteme eigene föderierte Schemata erstellt werden. Die Anwendungsersteller suchen sich aus den angebotenen Exportschemata die für sie relevanten Bestandteile aus und integrieren die betreffenden Exportschemata zu ihrem eigenem föderierten Schema. Anhand der beiden Skizzen 2.1 bzw. 2.2 für die beiden Architekturansätze können die beiden Konzepte der Schemaentwicklung leicht nachvollzogen werden.

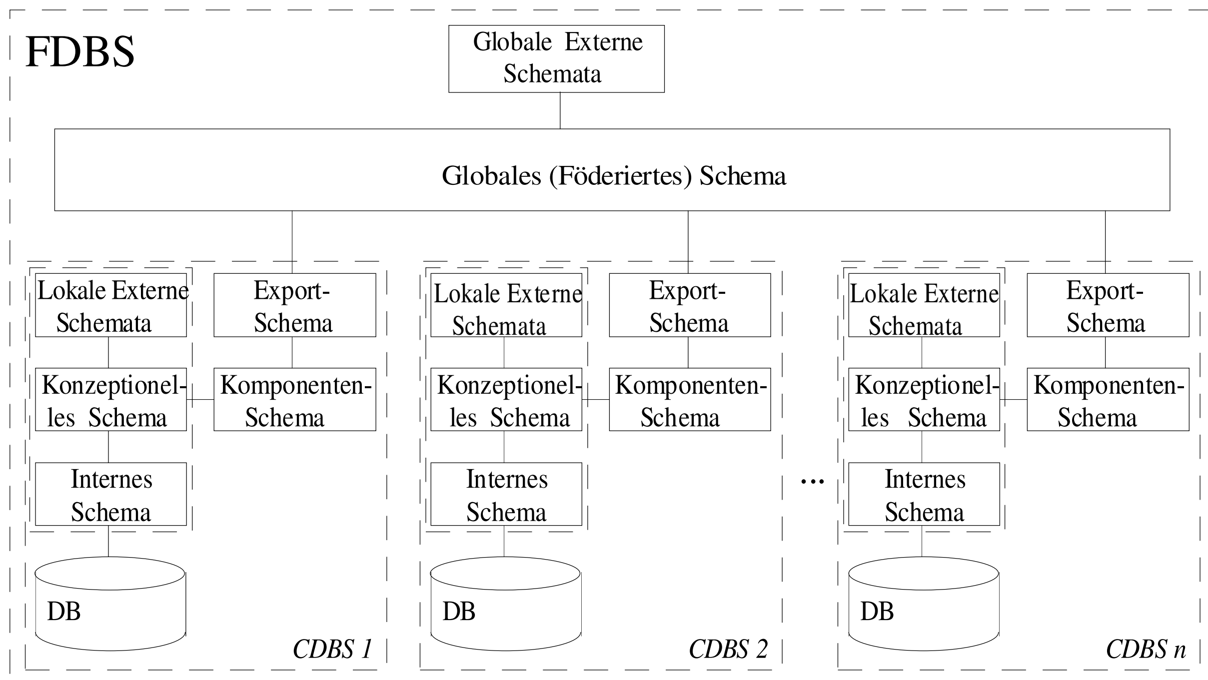

Abb. 2.2: FDBS-Architektur mit einem föderierten Schema - eng gekoppelter Ansatz

Die Abbildungen 2.1 bzw. 2.2 lassen auch die 5-Schichten-Schemaarchtektur für föderierte Datenbanksysteme nach [SL90] erkennen. Die Schritte der Erstellung der einzelnen Schemata sind hierbei jedoch aus Gründen der Übersichtlichkeit nicht explizit berücksichtigt worden.

Abbildung 2.2 zeigt ein eng gekoppeltes föderiertes Datenbanksystem mit nur einem föderierten Schema.

Föderierte Systeme mit nur einem Schema erleichtern die Konsistenzsicherung, da die Konsistenzbedingungen nicht für mehrere Schemata formuliert werden können bzw. müssen. Es kann in einem föderierten System mit mehreren föderierten Schemata der Fall eintreten, daß für semantisch gleiche Objekte in verschiedenen föderierten Schemata verschiedene oder auch gleichbedeutende Konsistenzbedingungen formuliert sind, die sich gegenseitig ergänzen oder widersprechen. Ein eng gekoppeltes System mit nur einem föderierten Schema erlaubt die semantisch einheitliche Interpretation der integrierten Daten und auch die eindeutige Formulierung von Konsistenzbedingungen. Dies folgt aus der Tatsache, daß ein einzelnes föderiertes Schema durch den oben erwähnten Abstimmungsprozess (Integration) entstanden ist, in dessen Verlauf zur Überwindung semantischer Heterogenität eine einheitliche Semantik für die Daten entsteht. Bei dieser Schemageneration oder eventuellen Modifikationen, wie die Hinzufügung von Konsistenzbedingungen, werden diese auch zwischen den Administratoren abgestimmt. Da bei einer lose gekoppelten Föderation kein Abstimmungsprozeß stattfindet, sondern jeder Interessent sein eigenes föderiertes Schema zusammenstellt, ergibt sich eine Vielzahl möglicher Wechselwirkungen solcher individuell festgelegten Konsistenzbedingungen. In lose gekoppelten föderierten Datenbanksystemen gibt es auch oftmals keine Bindung oder Beziehung zwischen den föderierten Schemata (weil sie nicht gewünscht ist) und deshalb ist eine Konsistenzsicherung, die mehrere solche föderierte Schemata überspannt, auch nicht sinnvoll.

2.3 Beispielszenarien für Föderationen von Datenbanksystemen

In diesem Abschnitt sollen verschiedene Szenarien entwickelt werden um zu zeigen, zu welchem Zweck föderierte Systeme aufgebaut werden können. Hier werden auch die verschiedenen Architekturkonzepte verwendet, da Anhand der Beispielszenarien die Einsatzgebiete der beiden Architekturen deutlich werden. Diese Szenarien dienen später zur Verdeutlichung von globalen Konsistenzbedingungen in föderierten Systemen.

2.3.1 Beispielszenarien für eine lose gekoppelte Föderation

Die lose gekoppelte Architektur eines föderierten Datenbanksystems soll am Beispiel einer unternehmensübergreifenden Föderation verdeutlicht werden. Voneinander unabhängige Firmen führen in diesem Fall eine Föderation ihrer Daten durch, um in bestimmten Geschäftsbereichen miteinander zu kooperieren. Hierbei könnte es sich z.B. um Fluggesellschaften und Flughafenbetreiber handeln, die ihre Leistungen besser vermarkten wollen. Diese föderierten Daten werden dann allen potentiellen Interessenten zugänglich

gemacht. Solche Interessenten sind z.B. Reisebüros. Informationssysteme für die Fluggäste können ebenfalls die bereitgestellten Daten nutzen.

Die Nutzer der Föderation (dies sind Anwendungsprogramme bzw. Anwendungsprogrammierer) haben verschiedene Sichten auf die Daten in den Komponentensystemen. Sie benötigen nur Zugriff auf die sie interessierenden Daten. In einer solchen Umgebung, wie sie in diesem Szenario beschrieben wird, ist es auch wahrscheinlich, daß ein Interessent die Föderation verläßt, die anderen Beteiligten jedoch das System weiterhin betreiben. Deshalb wird man hier den lose gekoppelten Ansatz wählen und jeder Interessent kann sich aus dem Informationsangebot sein eigenes Schema erstellen. Für autonome, mit Kommunikationsnetzen verbundene Systeme wird auch in [SL90] die Nutzung des lose gekoppelten Architekturansatz vorgeschlagen.

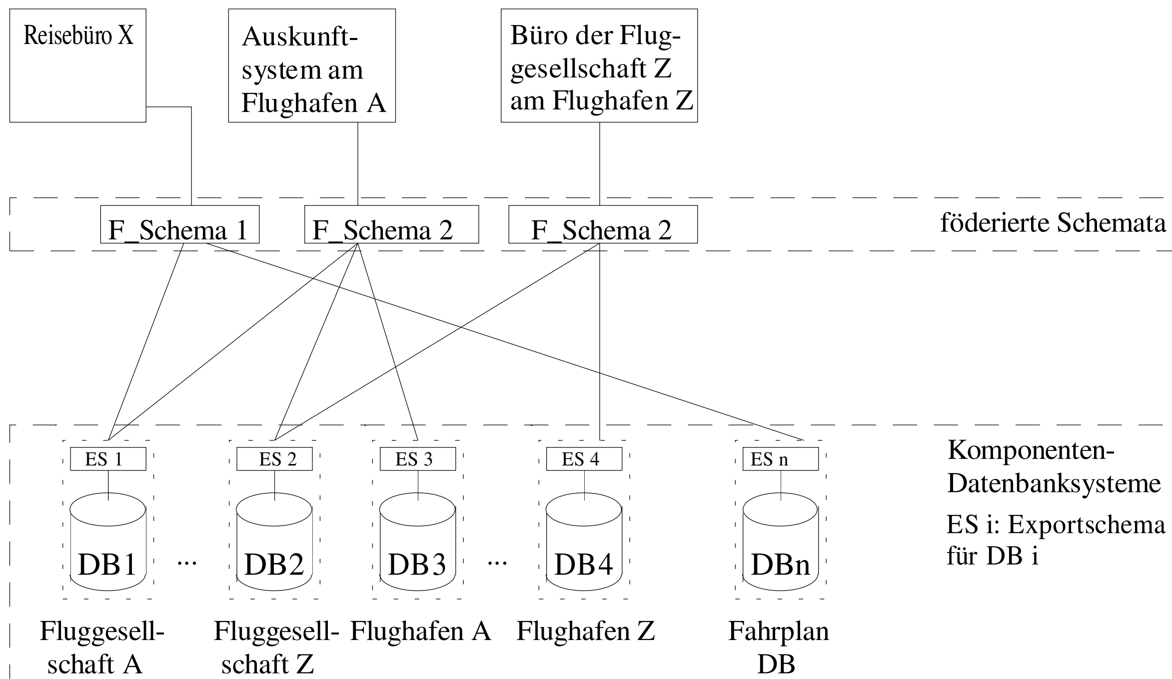

Abb. 2.3: Unternehmensübergreifende Föderation

Da die Interessenten weiterhin autonom bleiben wollen, ist eine globale Konsistenzsicherung in diesem System nicht erwünscht. Es gibt in diesem System keinen vereinigten Datenbestand, der wie ein zentralisiertes System konsistent zu erhalten ist (vgl. dazu 2.3.2). Deshalb wird dieses Beispiel im weiteren Verlauf der Arbeit auch nicht weitergehend betrachtet. Die beteiligten Komponentensysteme ihrerseits können und sollten jedoch ihre Daten konsistent erhalten und somit lokal begrenzte Konsistenzsicherung durchführen.

Ein weiteres Beispiel neben vielen anderen Einsatzmöglichkeiten föderierter Datenbanksysteme ist eine firmenübergreifende Konstruktionsdatenbank für größere Projekte. In [ABH+95] werden zwei Szenarien beschrieben, in denen Entwurfs- und

Konstruktionsprozesse durchgeführt werden. Charakteristisch für beide Szenarien ist, daß mehrere Personen ein Projekt bearbeiten. Es handelt sich außerdem um langwierige Prozesse, im deren Verlauf oftmals vorläufige Ergebnisse ausgetauscht werden. Wird für eine solche Konstruktionsaufgabe ein föderiertes Datenbanksystem benutzt, ergibt sich ein in der Architektur ähnlicher Aufbau des Systems wie im oben betrachteten Beispiel der Reisedatenbank - jedoch mit völlig anderer Charakteristik der Anwendungen.

Es ist besonders erwähnenswert, daß als lokale Anwendungen oder sogar als Komponentensystem an seinem solchen FD(B)S CAD-Systeme beteiligt sein können. Eine weitere Möglichkeit ist die Benutzung des Föderationsgedankens zur Verbindung mehrerer CAx-Systeme, denen unterschiedliche Datenbanksysteme unterliegen [BEK+94].

2.3.2 Beispielszenario für eine eng gekoppelte Föderation

In diesem Beispiel für eine eng gekoppelte Föderation von Datenbanksystemen wird eine Unternehmensdatenbank, die innerhalb einer Unternehmung organisiert werden soll, betrachtet. Ein Unternehmen besteht aus verschiedenen Abteilungen, wie Planung, Produktion, Verkauf und Verwaltung. In einigen dieser Abteilungen wurden dem Fortschritt bei der Entwicklung der Datenbanksysteme gemäß zu unterschiedlichen Zeitpunkten verschiedene (heterogene) Datenbanksysteme eingeführt und Datenbanken aufgebaut. Das Unternehmen produziert Geräte verschiedener Typen. Jedes Gerät hat eine Seriennummer und gemäß Kundenwünschen werden Extraausführungen gefertigt, so daß die Fertigung einzelner Geräte oder ganzer Baulose gleicher Geräte im voraus geplant werden muß. Das Unternehmen möchte nun die Datenbanksysteme der verschiedenen Abteilungen verbinden und einheitlichen Zugriff auf alle Unternehmensdaten ermöglichen. Da alle Daten zur Koordination innerhalb des Unternehmens konsistent sein müssen, ist in dem hier beschriebenen Szenario globale Konsistenzsicherung wichtig. Deshalb sollte in einer solchen Umgebung der eng gekoppelte Architekturansatz verwendet werden, da sich mit dieser Architektur eine globale Konsistenzsicherung am besten realisieren läßt. Die Abbildung 2.4 zeigt die eng gekoppelte Architektur des föderierten Systems. In dieser Abbildung wurden die Komponentendatenbanksysteme vereinfacht dargestellt, d.h. lokale, externe und Komponentenschemata wurden nicht berücksichtigt.

Die Föderierung einzelner Abteilungsdatenbanken zu einer Unternehmensdatenbank kann aus ganz unterschiedlichen Gründen forciert werden, und zwar:

- zur effizienten Unternehmenslogistik und -führung; um Unternehmensaktivitäten besser koordinieren zu können und
- zur zukünftig unternehmensweiten Vereinheitlichung der Datenhaltung durch spätere Migration der Abteilungsdaten.

Die Migration der Daten der einzelnen Abteilungen in ein zentrales System und die schrittweise Reduktion von Datenhaltungssystemen, wie von [RS94] beschrieben, dient dem Schutz vorangegangener Investitionen in die Datenhaltungssysteme der verschiedenen Abteilungen. Diese Migration wird sich i.d.R. nicht vollständig verwirklichen lassen, da hierzu ein System benötigt würde, das alle Anwendungen effizient ausführen kann. Dieses System müßte auch die speziellen Funktionalitäten und Datenmodellbesonderheiten der ursprünglichen Datenbanksysteme unterstützen.

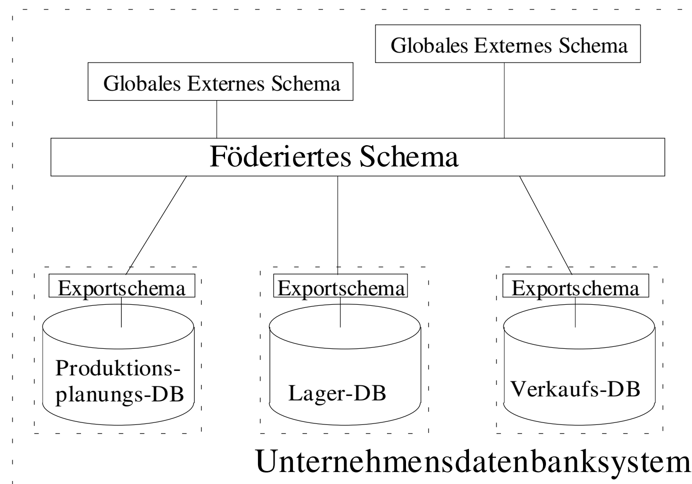

Abb. 2.4: Unternehmensweite Föderation mit eng gekoppelter Architektur

Durch die Wahl des eng gekoppelten Architekturansatzes (siehe Abb. 2.4) wird bei der Schemaintegration eine einheitliche Interpretation des Daten unternehmensweit vereinbart. In dem hier beschriebenen Szenario können die logistischen Bewegungen der Produkte durch Datenmigrationen nachgebildet werden, um ihre redundante Speicherung zu vermeiden. Eine globale Schlüsselbedingung ist dann nötig, da ein Produkt nur einmal im gesamten System auftreten darf. Es ist weiterhin nötig, dynamische Konsistenzbedingungen für die Wege der Daten innerhalb des Systems festzulegen. Ein Beispiel hierfür ist die folgende Bedingung: „Nur bereits produzierte oder gelagerte Produkte können verkauft werden."

Nun folgt die Beschreibung ausgewählter Tabellen bzw. eines Objektes aus den einzelnen Komponentensystemen. Hierzu wird angenommen, daß die Komponentensysteme des föderierten Datenbanksystems auf unterschiedlichen Datenmodellen basieren. Damit soll erreicht werden, daß verschiedenartige, lokale Konsistenzsicherungsmechanismen zur Überwachung globaler Konsistenzbedingungen herangezogen werden.

Die relationalen Schemafragmente werden mittels SQL-Notation beschrieben, das objektorientierte Beispiel mittels C++-Notation. Die Integrationsaspekte bei der Erstellung einer Föderation sollen in dieser Arbeit nicht näher betrachtet werden. Diese Sachverhalte werden in [BFN94] behandelt. Die Aufstellung der lokalen Schemafragmente und des

föderierten Schemafragmentes erfolgte in dieser Arbeit exemplarisch und ist nur als Beispiel zur Verdeutlichung von Konsistenzbedingungen in einem föderierten System zu sehen.

Die Produktionsplanungsdatenbank ist eine relationale Datenbank. Die in ihr befindliche Relation für die Exemplardaten wird durch den folgenden SQL-Befehl erstellt, der ein Teil des SQL-Skriptes zur Schemaerzeugung auf der relationalen Datenbank ist:

```
CREATE TABLE Exemplar (
 sernum          : CHAR(20) NOT NULL,
 typbez          : CHAR(15) NOT NULL,
 proddatum       : DATE,
 ...
 PRIMARY KEY (sernum, typbez)
);
```

In der durch diesen SQL-Befehl erzeugten Tabelle sind die zu produzierenden Exemplare verschiedener Gerätetypen abgelegt. Jedes Exemplar wird über eine Seriennummer in Verbindung mit einer Typenbezeichnung eindeutig identifiziert. Für jedes Exemplar wird darüber hinaus das Produktionsdatum gespeichert. Außerdem können noch Austattungsdaten und technische Parameter gespeichert werden, was jedoch in diesem Schemaentwurf nicht berücksichtigt werden soll.

Die Lagerdatenbank ist ebenfalls relational. Sie enthält eine Tabelle `Lagerung`, die durch den folgenden (unvollständigen) SQL-Befehl erzeugt werden kann:

```
CREATE TABLE Lagerung (
 seriennr        : VARCHAR2(22) NOT NULL,
 typ             : VARCHAR2(13) NOT NULL,
 eingdatum       : DATE,
 lagerplatz_id   : CHAR(10),
 ...
 PRIMARY KEY (SerienNr, Typ)
);
```

In der durch diesen Befehl erzeugten Tabelle ist die Zuordnung bereits produzierter Exemplare zu den Lagerplätzen enthalten. Es wird ein Lagereingangsdatum und der Lagerplatz verwaltet. Damit kann ein eingelagertes Gerät gefunden und seine Lagerdauer bestimmt werden.

Die Datenbank für den Verkauf ist objektorientiert und basiert auf der Programmiersprache C++. Der nun folgende Quelltext ist ein Ausschnitt aus einer Headerdatei, die z.B. in ONTOS zur Schemageneration herangezogen wird:

```
struct Datum {
 int             tag;
```

```
  int               monat;
  int               jahr;
 }
 class Auslieferung {
  private:
  char*             liefident;
  datum*            liefdatum;
  char*             typ;
  char*             sernum;
  float             preis;
  ...
  public:
  ...
  setLiefDatum      (datum* liefdatum);
  setTyp            (char* typ);
  setSerNum         (char* sernum);
  setPreis          (float preis);
  ...
  datum*            getLiefDatum;
  char*             getTyp;
  char*             getSerNum;
  float             getPreis;
  ...
  void              Lieferscheinerstellen(...);
  void              Lieferscheindruck(...);
  void              Verbuchen(...);
 }
```

Das föderierte Schema wird in der ODMG93-Notation beschrieben. Wie unter Zuhilfenahme von ODMG93 ein föderiertes Schema aufgebaut werden kann, ist in [BFN94] nachzulesen.

```
 interface produkt {
  extent produkte;
  key (seriennummer,typenbezeichnung);
  attribute  string<22>      seriennummer;
  attribute  string<15>      typenbezeichnung;
  attribute  date            produktionsdatum;
  attribute  string<10>      lagerplatzId;
  attribute  date            eingangsdatum;
  ...
  setEingDatum      (datum* edatum);
  setTyp            (char* typ);
  setSerNum         (char* sernum);
 }

 interface lager {
```

```
  extent gesamtlager;
  key (lagerid);
  attribute  string<15>      lagerId;
  attribute  set<produkt>    lagerbestand;
  ...
  int getBestandsHoehe (char* typ) ;
  void lagereEin (char* typ, char* sernum);
 }

interface lieferung
  extent verkauf;
  key (lieferungsnr);
  attribute  string<10>      lieferungsnr;
  attribute  date            lieferungsdatum;
  attribute  set<produkt>    liefergegenstand;
  attribute  float           gesamtpreis;
  ...
  setLiefDatum    (datum* liefdatum);
  setPreis        (float preis);
  ...
  datum*          getLiefDatum;
  float           getPreis;
  ...
  void       Lieferscheinerstellen(...);
  void       Lieferscheindruck(...);
 }
```

Auf der Grundlage dieser Schemafragmente soll exemplarisch die Aufstellung von Konsistenzbedingungen vorgenommen werden. Es werden dazu jeweils die Bedingung und die potentiell konsistenzverletzenden Methoden genannt.

Eine Konsistenzbedingung könnte lauten: „Das Lagereingangsdatum in der Lagerdatenbank muß älter als das Produktionsdatum sein.” Die Bedingung kann dann so formuliert sein:

```
produkt.produktionsdatum <= produkt.eingangsdatum
```

Die folgende Methode kann eine Inkonsistenz erzeugen:

```
produkt::setEingDatum (datum edatum)
```

Eine etwas andere Konsistenzbedingung ist dagegen diese: „Im Lager X müssen mindestens 20 Exemplare vom Typ Y vorhanden sein”. Die Bedingung unter Bezugnahme auf das föderierte Schema lautet hier:

```
lager.lagerId=X and lager::getBestandsHoehe(Y) = 20
```

Für diese Konsistenzbedingung kann der Aufruf der folgenden Methode eine Konsistenzverletzung verursachen:

```
lager::lagereAus (char* typ, char* sernum);
```

Diese beiden Bedingungen beschreiben Beziehungen zwischen Objekten und Tabellen, die in verschiedenen lokalen Datenbanken gespeichert sind. Im Kapitel 4 wird Anhand dieser Konsistenzbedingungen versucht, Mechanismen zur Konsistenzsicherung in föderierten Datenbanksystemen darzustellen. Es ist möglich, hier weitere Konsistenzbedingungen zu definieren. Anregungen dazu können die im Abschnitt 3.1 beschriebenen Konsistenzbedingungen bieten. So ist bereits für alle Produkte sowohl in der Produktionsdatenbank als auch in der Lagerdatenbank eine Schlüsselbedingung formuliert worden.

3 Grundlagen der Konsistenzsicherung in DBSen

Für den Nutzer eines Datenbanksystems (DBS) ist es wichtig, daß die Daten in den Datenbanken korrekt sind, d.h. gewisse Bedingungen erfüllen. Es gibt jedoch mehrere Gründe für das Auftreten inkorrekter Daten: unautorisierte Datenmanipulation, gleichzeitiges Arbeiten mehrerer Nutzer mit denselben Daten, Hard- bzw. Softwarefehler und semantisch fehlerhafte Datenzustände infolge von Datenmanipulationen. Fehler, die durch die drei erstgenannten Ursachen entstehen können, sollten jedoch in der Regel durch das Datenbankmanagementsystem verhindert oder behoben werden. Bei Hard- oder Softwarefehlern muß eine Reparatur der Hardware oder ein Austausch der betroffenen Hardware oder Software erfolgen. Die Verhinderung der letztgenannten semantischen Fehler bzw. semantisch fehlerhaften Datenbankzustände wird als Integritäts- oder Konsistenzsicherung bezeichnet.

Eine Datenbank ist in einem *konsistenten Zustand*, wenn keine der auf der Datenbank definierten *Konsistenzbedingungen* verletzt ist [Vos94]. Konsistenzbedingungen werden in der Literatur auch häufig als *Integritätsbedingungen* oder *Constraints* bezeichnet.

Konsistenzbedingungen dienen zur Formulierung von Korrektheitskriterien für Daten. Es gibt eine Vielzahl solcher Bedingungen. Im Abschnitt 3.1 wird zunächst eine Einteilung der wichtigsten Formen von Konsistenzbedingungen für relationale und objektorientierte Datenbanksysteme durchgeführt. Anhand je eines Beispielsystemes - ORACLE 7 als Vertreter der relationalen Datenbanksysteme und ONTOS DB 3.0 als Vertreter der objektorientierten Datenbanksysteme - soll die Unterstützung der Formulierung dieser Konsistenzbedingungen in existierenden Systemen gezeigt werden. Im Abschnitt 3.2 werden die für die Konsistenzsicherungsaufgaben vielseitig nutzbaren ECA-Regeln vorgestellt. Im Anschluß daran wird im Abschnitt 3.3 die Bedeutung von Konsistenzbedingungen in föderierten Systemen dargelegt.

3.1 Konsistenzbedingungen in zentralen Systemen

Konsistenzsicherung in zentralen Systemen kann auf verschiedene Weise durchgeführt werden: entweder durch das Datenbankmanagementsystem selbst oder durch die auf dem Datenbanksystem laufenden Anwendungen. Wenn die Konsistenzsicherung in zentralisierten Datenbanksystemen ein zum Datenbankmanagementsystem gehörender Monitor übernimmt, können alle Konsistenzbedingungen einheitlich, d.h. unabhängig von den vorhandenen Anwendungen vereinbart und überwacht werden. Außerdem werden so die Anwendungsprogramme von Konsistenzsicherungsaufgaben entlastet. Deshalb gibt es in

heutigen zentralisierten Datenbanksystemen einen Monitor, der die Einhaltung von Konsistenzbedingungen prüft und überwacht.

In den bestehenden Datenmodellen für Datenbanken gibt es prinzipiell drei verschiedene Arten von Konsistenzbedingungen:

- statische,
- transitionale und
- dynamische Konsistenzbedingungen.

Statische Konsistenzbedingungen beschränken die Menge der durch die Festlegungen des betreffenden Schemas möglichen Zustände. Sie beschreiben zum Beispiel Wertebereiche für Attribute. *Transitionale* (oder auch *halbdynamische*) *Konsistenzbedingungen* schränken die möglichen Zustandsübergänge ein. Es wird immer genau ein Paar von möglichen Zuständen, ein Zustandsübergang, festgelegt. *Dynamische Konsistenzbedingungen* schränken (als Verallgemeinerung der transitionalen Bedingungen) die möglichen *Zustandsfolgen* ein. Sie werden auch als *temporale Bedingungen* bezeichnet.

3.1.1 Konsistenzbedingungen in relationalen Datenbanksystemen

Im relationalen Datenmodell sollen nun die statischen Konsistenzbedingungen betrachtet werden. Folgende statische Konsistenzbedingungen gibt es in relationalen Datenbanken:

1. Domain- oder Attributbedingungen,
2. Tupelbedingungen,
3. Relationenbedingungen sowie
4. Referentielle Bedingungen.

1. *Domain- oder Attributbedingungen* schränken die Werte ein, die ein Attribut in einem Tupel einer Relation annehmen kann. Hierzu gehören:

- die Festlegung von Grenzen für die Werte eines numerischen Attributes,
- die Festlegung einer Menge möglicher Werte für ein alphanumerisches Attribut und
- die `NOT NULL`-Bedingung.

Diese Bedingungen sind immer auf ein Attribut eines Tupels beschränkt. Domain- oder Attributbedingungen für numerische und alphanumerische Attribute können recht einfach durch Beispiele verdeutlicht werden. Eine Attributbedingung für ein numerisches Attribut ist z.B. die folgende: Für das numerische Attribut `Alter` einer Relation `Person` kann (mit hoher Wahrscheinlichkeit) der Wert auf positive Zahlen kleiner als 130 begrenzt werden. Für ein alphanumerisches Attribut kann eine Konsistenzbedingung durch das folgende kleine Beispiel verdeutlicht werden: Will man in einem alphanumerischen Feld eine Information über eine Auswahl (aus Programmiersprachen als Aufzählungstyp bekannt), z.B. ob eine Person einen Führerschein besitzt, speichern, können die möglichen Eintragungen `ja`, `nein` oder `unbekannt` lauten. Die `NOT NULL`-Bedingung besagt, daß das betreffende Attribut einen Wert haben *muß*. `NULL` bedeutet, daß *kein* Wert existiert oder bekannt ist. Dieser Attributzustand wird im weiteren Verlauf der Arbeit als Nullwert bezeichnet.

2. *Tupelbedingungen* betreffen einzelne Tupel innerhalb einer gegebenen Relation. Eine solche Bedingung umfaßt immer mehrere Attribute ein- und desselben Tupels. In der Regel werden die Werte der Attribute verglichen. Eine Tupelbedingung ist z.B. „Das Jahr der Einschulung einer Person muß mindestens 6 Jahre nach dem Geburtsjahr liegen".

3. *Relationenbedingungen* wirken auf die Menge *aller* Tupel einer Relation. Folgende Relationenbedingungen sind verbreitet:

- Schlüsselbedingungen,
- Aggregatbedingungen und
- Rekursive Bedingungen.

Schlüsselbedingungen fordern die Schlüsseleigenschaft für ein oder mehrere Attribute eines Relationenschemas. Über einen Schlüssel kann jedes Tupel der Relation eindeutig identifiziert werden, d.h. ein Schlüssel enthält jeden Wert nur einmal. Bei mehrwertigen Schlüsseln können Teile des Schlüssels in mehreren Tupeln den gleichen Wert besitzen. In einer *Aggregatbedingung* wird eine Bedingungen für ein Aggregat formuliert. Ein Beispiel dafür ist die Beschränkung der Summe der Werte eines Attributes in jedem Tupel durch einen Grenzwert. Dies könnte konkret so aussehen: Das Attribut `Gehalt` ist in einer Tabelle `Angestellte` enthalten. Die Summe der Gehälter aller Angestellten soll ein Budget, z.B. 100000 DM, nicht überschreiten. *Rekursive Bedingungen* können zur Forderung des transitiven Abschlusses einer Relation dienen. So kann eine Datenbank für Zugverbindungen folgende rekursive Bedingung einhalten: „Jeder Ort ist von jedem Ort aus erreichbar".

4. *Referentielle Bedingungen* spezifizieren semantische Verbindungen zwischen Paaren von Relationen. Typisch sind hier die Definitionen von Fremdschlüsseln mit einem oder mehreren Attributen mit bzw. ohne Nullwerte.

Attributbedingungen, Tupelbedingungen und Bedingungen auf Relationenebene können im Relationenmodell auch *dynamische* Bedingungen sein. Dynamische Konsistenzbedingungen können mittels Trigger realisiert werden.

Das Triggerkonzept relationaler Datenbanksysteme

Die in den relationalen Datenbanksystem vorhandenen Trigger ermöglichen eine automatisierte Konsistenzprüfung. Es ist jedoch Voraussetzung, daß die auslösenden Datenbankoperationen eine Integritätsprüfung oder sogar ein Undo bzw. Rollback in SQL unterstützen.

Der grundsätzliche Aufbau eines Triggers beinhaltet die auslösende Datenbankoperation, eine logische Bedingung sowie eine Anweisung, die beim Auftreten der Datenbankoperation und Erfüllung der Bedingung ausgeführt werden soll:

```
TRIGGER ON {INSERT/UPDATE/DELETE}
IF NOT (bedingung) THEN UNDO/ROLLBACK
```

Die als Auslöser in Frage kommenden Datenbankoperationen sind also `INSERT, UPDATE` sowie `DELETE`. In der Bedingung können Beziehungen zwischen den schon modifizierten und den noch unveränderten Datenwerten beschrieben werden. Dies unterstützt die Formulierung transitionaler Konsistenzbedingungen. Hierzu muß jedoch besonders bei Modifikationen von Daten auf die alten und die neuen Daten intern zugegriffen werden können.

Die Bedingung beschreibt einen gültigen Datenbankzustand. Ist die Bedingung beim Eintreten der vordefinierten Datenbankoperation nicht erfüllt, wird automatisch die dem Schlüsselwort `THEN` folgende Anweisung ausgeführt. In diesem einfachen Trigger ist lediglich die Zurücksetzung der gerade durchgeführten Datenbankoperation vorgesehen [Vos94].

Konsistenzsicherungsmechanismen im relationalen Datenbanksystem ORACLE

Die Verwendung der in den vorangegangenen Absätzen beschriebenen Konsistenzbedingungen einschließlich der Trigger werden von den kommerziellen relationalen Datenbanksystemen in unterschiedlicher Weise unterstützt. Das relationale System ORACLE unterstützt alle der zuvor beschriebenen Konsistenzbedingungen, einige Bedingungen gibt es jedoch erst seit der ORACLE-Version ORACLE 7 [Stü93].

ORACLE kennt dazu zwei verschiedene Realisierungsmittel für Konsistenzbedingungen: *deklarative* und *prozedurale* Konsistenzbedingungen. Die deklarativen Konsistenzbedingungen

werden bei der Definition der Tabellen verwendet. Die nun folgende Tabelle enthält eine Aufstellung der deklarativen Konsistenzbedingungen:

Konsistenzbedingung	Bedeutung
`primary key`	definiert die Primärschlüsseleigenschaft für ein oder mehrere Spalten
`unique`	definiert Schlüsseleigenschaft für ein oder mehrere Spalten der Tabelle
`check`	erlaubt die Definition einer Bedingung für ein Attributwert
`foreign key/ references`	definiert eine Fremdschlüsselbeziehung; ist im Zusammenhang mit Primärschlüssel zur Realisierung von referenziellen Bedingungen nötig
`not NULL/NULL`	erlaubt bzw. verbietet Nullwerte
`default`	ermöglicht die Festlegung eines Standardwertes für ein Attribut
`on delete cascade`	Sicherung der referentiellen Integrität bei Löschoperationen

Tabelle 3.1: Deklarative Konsistenzbedingungen in ORACLE 7

Die im letzten Abschnitt besprochenen Trigger werden in ORACLE auch als prozedurale Konsistenzbedingungen bezeichnet. Trigger werden zusätzlich zu den deklarativen Konsistenzbedingungen definiert. Ein ORACLE-Trigger wird folgendermaßen vereinbart oder modifiziert:

```
CREATE/REPLACE TRIGGER <Triggername>
AFTER/BEFORE
INSERT/UPDATE/DELETE OF <Menge von Spalten>
ON <Tabllenname>
FOR EACH ROW
WHEN <Bedingung>
plsql-block
```

Mit der `CREATE TRIGGER`-Anweisung wird ein Trigger erzeugt, während `REPLACE TRIGGER` einen bereits existierenden Trigger ersetzt. Ein ORACLE-Trigger überwacht Manipulationen an Daten und erkennt `INSERT`, `DELETE` und `UPDATE`-Operationen auf den Spalten einer Tabelle. Beim Eintreten einer dieser Operation kann eine Bedingung ausgewertet werden und wenn diese Bedingung erfüllt ist, wird ein Programm ausgeführt, das in der ORACLE eigenen Programmiersprache PL/SQL als in sich abgeschlossener Block formuliert sein muß. Die Bedingung eines ORACLE-Triggers kann eine Anfrage enthalten, deren Ergebnis innerhalb der Bedingung zu einem Vergleich herangezogen wird. Teilbedingungen können innerhalb einer Bedingung mit den logischen Operationen `AND` und `OR` verknüpft

werden. Der volle Umfang einer solchen Bedingung wird in ORACLE-SQL Handbüchern, wie z.B. [ORA92], unter dem Stichwort `condition` erklärt .

3.1.2 Konsistenzbedingungen in objektorientierten Datenbankmanagementsystemen

Nach [JQ92] können Konsistenzbedingungen in objektorientierten Systemen in zwei Gruppen, *intraobjektive* und *interobjektive* Konsistenzbedingungen, eingeteilt werden. Intraobjektive Bedingungen betreffen immer einzelne Objekte. Dazu gehören die Bedingungen auf Attribut- und Objektebene. Interobjektive Bedingungen beschreiben Beziehungen zwischen mehreren Objekten. Im einzelnen gibt es folgende intraobjektive Konsistenzbedingungen:

- Bedingungen auf Attributebene,
- Bedingungen auf Objektebene,
- Extensionsbedingungen und
- Referenzielle Bedingungen.

Die Bedingungen auf *Attributebene* sind mit den Attributbedingungen des relationalen Modells vergleichbar. So gibt es z.B. auch die `NOT NULL`-Bedingung in objektorientierten Datenbankmanagementsystemen.

Die Bedingungen auf *Objektebene* betreffen jeweils einzelne Objekte. Sie können z.B. Beziehungen zwischen verschiedenen Attributen eines Objektes sein. Sie sind mit den Tupelbedingungen im relationalen Modell vergleichbar.

Die *Extensionsbedingungen* betreffen die Extension, d.h. alle Objekte einer Klasse. Sie sind den Relationenbedingungen vergleichbar. Die Schlüsselbedingung ist schon implizit durch Verwendung von Objektidentitäten garantiert. Da diese Identitäten jedoch dem Datenbanknutzer nicht zugänglich sind, wird die Definition eigener Schlüsselattribute von objektorientierten Datenbankmanagementsystemen unterstützt.

Die *referenziellen Bedingungen* in objektorientierten Datenbanksystemen sind folgende:

- Referentielle Integrität,
- die existentielle Abhängigkeit entlang einer Referenz und
- die Festlegung von inversen Referenzen.

Die referentielle Integrität ist bereits implizit durch Referenzmechanismen in objektorientierten Datenbanksystemen realisiert. Die existentielle Abhängigkeit entlang einer Referenz kann auch durch Referenzmechanismen gewährleistet werden. Die Festlegung einer Inversreferenz kann z.B. in ONTOS explizit gefordert werden.

Integritätsbedingungen in objektorientierten Datenbanksystemen können auch transitional oder dynamisch sein. Im folgenden Abschnitt soll nun am Beispiel des objektorientierten Systems ONTOS die Unterstützung deklarativer Konsistenzbedingungen durch objektorientierte Datenbanksysteme beschrieben werden.

Konsistenzbedingungen im objektorientierten Datenbanksystem ONTOS

ONTOS ist ein C++-basiertes, objektorientiertes Datenbanksystem. In ONTOS erfolgt die Generation der Datenbankschemata mittels eines speziellen Werkzeuges auf der Grundlage der in C++ üblichen Headerdateien. Bei der Schemaerzeugung kann auch eine Steuerdatei, in der deklarative Konsistenzbedingungen aufgeführt sein können, berücksichtigt werden. In dieser Datei ist die Verwendung folgender Schlüsselwörter zur Festlegung von Konsistenzbedingungen möglich [ONT94]:

Schlüsselwort	Bedeutung
`is unique`	fordert die Schlüsseleigenschaft für ein Attribut
`is required`	semantisch gleichwertig zum `NOT NULL` anderer DBSe
`is inverse`	definiert Inversreferenz oder Rückreferenz

Tabelle 3.2: Konsistenzbedingungen des objektorientierten Datenbanksystem ONTOS

Die Verwendung dieser Konstrukte wird nachfolgend an einem Beispiel demonstriert:

Beispiel 3.6: Eine Klasse `PERSON` ist definiert und es sollen verschiedene Konsistenzbedingungen für ihre Objekte gelten. Eine Person soll immer einen Namen haben und dieser Name soll (nicht ganz realitätsnah) eindeutig sein. Die Klasse `PERSON` besitzt eigene Objekte, d.h. sie hat eine Extension. Es werden die Referenzen `vater` und `sohn` als invers zueinander vereinbart. In der Steuerdatei werden diese Bedingungen folgendermaßen deklariert:

```
class PERSON has extension
PERSON::name is required is unique
PERSON::vater inverse PERSON::sohn
```

In der ersten Zeile dieses Ausschnittes aus einer ONTOS-Steuertdatei wird die Existenz der Extension der Klasse `PERSON` festgelegt. Die zweite Zeile fordert mit `is required`, daß eine Person nicht namenlos sein kann. Sie legt auch die Einzigartigkeit des Namens einer Person durch die Angabe von `is unique` fest. Die Inversreferenz zwischen `vater` und `sohn` wird in der letzten Zeile erklärt.

3.2 Das ECA-Regel Konzept

Die Aktivität eines aktiven Datenbanksystems (ADBS) basiert auf der Fähigkeit, bestimmte vordefinierte Umweltsituationen zu erkennen und darauf mit entsprechenden, ebenfalls vordefinierten, Aktionen zu reagieren. Hierzu wurde das Konzept der ECA-Regeln zur Realisierung aktiver Funktionalität von Datenbanksystemen im HIPAC-Projekt [DBB+88] vorgeschlagen. Dieses Konzept etablierte sich und wurde in mehreren Forschungsprototypen implementiert. Zwei bekannte Beispiele für die Arbeit an aktiven Datenbanksystemen sind REACH[2] [BZBW95] und SAMOS[3] [GGD94]. ECA-Regeln sollen im Verlauf dieser Arbeit zur Konsistenzsicherung in föderierten Datenbanksystemen herangezogen werden. Dieser Abschnitt beschäftigt sich nun mit der Vorstellung des ECA-Regel Konzeptes.

ECA-Regeln in aktiven Datenbanksystemen ermöglichen *Reaktionen* auf vordefinierte *Situationen*. Eine *Situation* wird durch ein *Ereignis* und eine *Bedingung* beschrieben.

Eine ECA-Regel besteht also aus:

- **E**reignis (**e**vent),
- **B**edingung (**c**ondition) und
- (Re)**A**ktion (**a**ction).

Die Abarbeitung einer ECA-Regel geschieht nun folgendermaßen: Das vordefinierte Ereignis wird signalisiert. Wie dies realisiert sein kann, wird in diesem Abschnitt zu einem späteren Zeitpunkt beschrieben. Nach der Erkennung des Ereignisses wird überprüft, ob die Bedingung der Regel erfüllt ist. Ist die Bedingung der Regel erfüllt, wird die Aktion ausgeführt. Diese zeitliche Beziehung zwischen Ereignis und Bedingung und die Beziehung zwischen Bedingung und Aktion wird durch zwei Kopplungsmodi, die sogenannten EC- und CA-Kopplungsmodi, beschrieben. So kann z.B. festgelegt werden, ob die Bedingung unmittelbar nach dem Eintreten des Ereignisses oder erst am Ende der zugehörigen Transaktion überprüft werden soll (siehe [Buc94]).

In den nun folgenden Abschnitten werden die Bestandteile einer ECA-Regel detailliert beschrieben. Ereignis, Bedingung und Aktion werden dabei in jeweils einem eigenen Abschnitt betrachtet.

---

[2]REACH: REal-time ACtive and Heterogeneous mediator System; Projekt der TH Darmstadt, FB Informatik, Datenverwaltungssysteme 1

[3]SAMOS : Swiss Active Mechanism-based Object oriented database System; Universität Zürich, Institut für Informatik, Forschungsbereich Datenbanktechnologie

Ereignisse

Ein *Ereignis* zeigt einen Zeitpunkt während einer Programmabarbeitung an. Dies kann ein Aufruf einer Methode in einem objektorientierten System sein. Im Zuge der Forschungsarbeit an aktiven Systemen wurden verschiedene Ereignistypen zusammengestellt (weitere Details sind in [CM93, GD93] nachzulesen). Allgemein werden *primitive* und *zusammengesetzte* Ereignisse unterschieden. Primitive Ereignisse zeigen entweder Beginn oder Ende einer Operation (z.B. Beginn oder Ende einer Transaktion) an, oder betreffen eine bestimmte Uhrzeit (wie z.B. 9.00.00 Uhr). Darüber hinaus werden in [GD93] periodische Zeitereignisse beschrieben, d.h. das Eintreten des Ereignisses wird in immer wiederkehrenden Zeitabständen signalisiert.

Die Signalisierung eines Methodenereignisses (d.h. die Ausführung einer Operation in einem objektorientierten Datenbanksystem) kann vor oder nach der Abarbeitung des Methodenquelltextes erfolgen. Diese beiden Zeitpunkte werden als `before` bzw. `after` bezeichnet. Die Ausführung der in relationalen DBS existierenden Datenmanupulationsoperationen wie `INSERT`, `UPDATE` und, `DELETE` sind vergleichbar mit Methodenereignissen, da äquivalente Datenmanipulationen in den objektorientierten Datenbanksystemen mittels Methoden durchgeführt werden müssen.

Primitive Ereignisse können mittels sogenannter Ereignisoperatoren zu zusammengesetzten bzw. kompositen Ereignissen verknüpft werden. Die folgende Tabelle zeigt die wesentlichen Ereignisoperatoren existierender Prototypen aktiver Datenbanksysteme auf. In der Tabelle 3.3 wird die Notation des aktiven Datenbanksystem REACH verwendet (vgl. [Deu94, Tür94]):

Operator	Bezeichnung	Komplexes Ereignis wird angezeigt, wenn:
`E1 or E2`	logisches ODER	eines der beiden primitiven Ereignisse `E1` oder `E2` signalisiert wird.
`E1 then E2`	Sequenz	`E1` zeitlich vor `E2` eintritt.
`E1 and E2`	logisches UND	beide Ereignisse ohne Beachtung der Reihenfolge eingetreten sind.
`NOT E2 in E1, E3`	Negation	`E2` zwischen `E1` und `E3` nicht eintritt.
`n times E2 in E1,E3`	History-Operator	`E2` zwischen `E1` und `E3` genau n-mal eintritt.
`all E2 in E1,E3`	Hüllenoperator (Closure)	in dem Intervall, das durch das Auftreten von `E1` bzw. `E3` begrenzt wird, `E2` ein- oder mehrmalig eintritt.

Tabelle 3.3: Standartoperatoren zur Definition zusammengesetzter Ereignisse

Die Negation, der Historie- und der Hüllenoperator sind an Ereignisintervalle gebunden. Diese Intervalle werden durch das Eintreten der sie begrenzenden Ereignisse beschrieben. Eine solche Betrachtungsweise wird von manchen Anwendungen gefordert. Mit der Negation kann man feststellen, ob in einem Ereignisintervall ein interessierendes Ereignis eingetreten ist. Soll das ein- oder mehrmalige Eintreten eines Ereignisses bezüglich eines Intervalls nur einmal angezeigt werden, kann der Hüllenoperator verwendet werden. Soll dagegen ein genau n-mal eintretendes Ereignis bezüglich eines Intervalls erkannt werden, erfolgt dies durch Verwendung des History-Operators, der das n-malige Eintreten nur einmal anzeigt.

Bedingungen

Eine Bedingung nimmt stets Bezug auf einen Datenbankzustand. Mittels einer Bedingung lassen sich sowohl korrekte als auch fehlerhafte Datenbankzustände beschreiben. In der Regel gibt es zwei Möglichkeiten, Bedingungen zu formulieren [DBB+88, GHW95]:

- Bedingung als *Prädikat* oder
- Bedingung als *Datenbankanfrage (Query).*

Wird als Bedingung ein Prädikat verwendet, erfolgt die Auswertung der Bedingung durch die Zuordnung eines booleschen Wertes. Die Bedingung ist genau dann erfüllt, wenn das Prädikat den Wahrheitswert `TRUE` besitzt. Im Gegensatz dazu wird bei der Bedingung als Datenbankanfrage diese Anfrage durchgeführt. Wenn das Ergebnis der Anfrage nicht leer ist, ist die Bedingung erfüllt. Das Ergebnis selbst kann dem Aktionsteil der ECA-Regel zur Weiterverwendung zur Verfügung gestellt werden.

Aktionen

Aktionen sind Anweisungen, die beim Eintreten o.g. Situationen auf der Datenbank ausgeführt werden sollen. Es ist zu beachten, daß auch durch Aktionen weitere Regeln ausgelöst werden können, wenn die Ausführung der als Aktion definierten Methode einer anderen Regel als Ereignis signalisiert wird (oder die in der Aktion aufgerufene Methode weitere Methoden aufruft). In relationalen Datenbanksystemen sind Aktionen meist auf typische Datenbankoperationen, wie z.B. `UPDATE`, `INSERT` und `DELETE` von Tupeln beschränkt. In objektorientierten Datenbanksystemen werden solche Operationen jedoch mittels Methoden durchgeführt. Daher liegt es nahe, beliebige Methodenaufrufe als Aktionen zuzulassen.

Die Bestandteile einer ECA-Regel wurden in den zurückliegenden Abschnitten beschrieben, so daß jetzt der volle Funktionsumfang einer ECA-Regel bekannt ist. Auf dieser Grundlage wird nun ein Vergleich der ECA-Regeln zu den Triggern relationaler Systeme durchgeführt.

ECA-Regeln und die Trigger des relationalen Datenbanksystems ORACLE

Die Trigger des relationalen Datenbanksystems ORACLE sind fast genauso mächtig wie ECA-Regeln. Es gibt bei diesem Trigger jedoch Einschränkungen bezüglich der Menge der auslösenden Datenbankoperationen und der zu startenden Aktionen. Eine universelle ECA-Regel kann auf allen möglichen Operationen definiert werden und alle durch Methodenaufrufe realisierbare Aktionen veranlassen. Die ORACLE-Trigger erlauben dagegen als Ereignis nur bestimmte SQL-Operationen und als Aktion nur einen Programmblock in der ORACLE-eigenen Programmiersprache PL/SQL. Die Bedingung eines ORACLE-Triggers kann z.B. auch eine Datenbankanfrage enthalten, deren Ergebnis zu einem Vergleich innerhalb der Bedingung herangezogen wird.

Zur Erkennung der in ECA-Regeln als Ereignis zugelassenen beliebigen Methodenaufrufe objektorientierter Systeme bedarf es eines besonderen Verfahrens. Deshalb wird im folgenden Abschnitt ein Konzept zur Detektion von Methodenaufrufen vorgestellt.

Das Method-Wrapping

Das „Method-Wrapping” wird verwendet, um Ereignisse, die durch Anweisungen ausgelöst werden (Methoderereignisse der objektorientierten Datenbanksysteme) zu erkennen. Dazu muß der Aufruf von beliebigen Methoden in Anwendungsprogrammen vom Datenbankmanagementsystem erkannt werden. Dies geschieht, indem jeweils direkt vor und direkt nach der ursprünglichen Methode spezieller Programmcode eingefügt wird. Das Prinzip, Methoden mit zusätzlichen Programmcode sozusagen „einzupacken”, wird „Method-Wrapping” genannt. Die zu „wrappende” Methode erhält hierzu einen neuen, generierten Namen. Es existiert weiterhin eine Methode mit dem ursprünglichen Namen, sie ruft jedoch die zusätzlich eingefügten Codes und die umbenannte ursprüngliche Methode auf [Deu94]. Ein Beispiel soll nun zeigen, wie eine Methode durch einen Precompiler modifiziert wird:

Beispiel 3.7: Die folgende vereinfacht notierte C++-Methode c::m() sei gegeben:

```
c::m()
{
 <Anweisungen>
}
```

Diese Methode wird durch den Preprozessor zunächst verändert in:

```
c::m_WRAPPED()
{
 <Anweisungen>
}.
```

Desweiteren wird noch eine neue Methode mit dem ursprünglichen Namen der bearbeiteten Methode generiert:

```
c::m()
{
     <signalisiere before-Ereignis>
     <Aufruf von c::m_WRAPPED()>
     <signalisiere after-Ereignis>
}
```

Die zur Umbenennung der ursprünglich vorhandenen Methoden benötigten neuen Namen werden nach einem logisch konstruierten Schema während des Vorübersetzungsprozesses generisch erzeugt. In dem Beispiel wurde nur ein beispielhafter Name verwendet, der die Umbenennung der ursprünglichen Methode symbolisiert. Die Nutzung eines Vorübersetzers ist in objektorientierten Datenbanksystemen gängige Praxis, da die (Haupt)Übersetzung durch den Compiler der Hostsprache erfolgt. Natürlich ist dieses „Method-Wrapping" nicht an objektorientierte Programmiersprachen oder Datenbanksysteme gebunden. Es kann vielmehr prinzipiell in jeder prozeduralen Programmiersprache, die zur Anwendungsgestaltung verwendet wird, genutzt werden.

3.3 Konsistenzbedingungen in föderierten Systemen

In föderierten Datenbanksystemen gibt es mehrere lokale Datenbanksysteme, die insgesamt eine semantisch vereinigte globale Datenbank bilden. Wenn in diesem System Konsistenzsicherung betrieben werden soll, so ist zunächst zu überlegen, welche Arten der im Abschnitt 3.1 beschriebenen Konsistenzbedingungen auf mehrere Systeme bezogen auftreten können bzw. für welche Bedingungen eine systemübergreifende Realisierung unmöglich bzw. unzweckmäßig ist.

Da in föderierten Datenbanksystemen oftmals eine Zusammenfassung semantisch gleichwertiger Objekte erfolgt, existieren über mehrere Datenbanken verteilte Mengen solcher Objekte. Hieraus ergibt sich die Notwendigkeit von datenbankübergreifenden interobjektiven Konsistenzbedingungen, was bei der Überwachung derartiger Bedingungen zu beachten ist. Die Existenz datenbankübergreifender intraobjektiver Konsistenzbedingungen hängt von der Möglichkeit der Verteilung einzelner Objekte über mehrere Komponentensysteme ab.

In diesem Abschnitt werden nun die aus Abschnitt 3.1 bekannten Konsistenzbedingungen bezüglich der Verteilungsmöglichkeit näher untersucht. Mögliche globale Konsistenzbedingungen sind:

- *Schlüsselbedingungen:* Es gibt eine Menge semantisch gleichartiger Objekte, die über verschiedene Komponentensysteme verteilt vorliegt. Für eine solche Menge als Ganzes muß die Schlüsseleigenschaft bezüglich ausgewählter Attribute unterstützt werden.

  Beispiel 3.1: Sollen logistische Prozesse in einer Unternehmensdatenbank (wie im Abschnitt 2.3 beschrieben) mittels Datenmigrationen nachvollzogen werden, ergibt sich daraus die Forderung nach einer systemübergreifenden Schlüsselbedingung, da ein Objekt in der Realität nicht an zwei Orten gleichzeitig existieren kann. Es kann mit einer solchen Bedingung kontrolliert werden, ob Datenmigrationen korrekt erfolgt sind.

- *Aggregatbedingungen*, wie die Beschränkung der Summe der Werte eines Attributes äquivalenter Objekte in mehreren Systemen, wenn die betreffende Objektmenge über mehrere Systeme verteilt vorliegen.

  Beispiel 3.2: Angestellte einer Firma sind mit ihren Gehältern in Abteilungsdatenbanken gespeichert und es wird angenommen, daß es keine extra Personaldatenbank gibt. Die Kontrolle, daß die Gesamtsumme der Gehälter ein entsprechendes Budget nicht überschreitet, ist dann eine komponentensystemübergreifende Aggregatbedingung.

- *Rekursive Bedingungen:* Sie können zur Forderung des transitiven Abschlusses einer Menge von in verschiedenen Datenbanksystemen befindlichen Datenelementen dienen.

  Beispiel 3.3: Eine verteilte oder föderierte Datenbank für Reiseverbindungen enthält folgende Bedingung: „Jeder Ort ist von jedem Ort aus erreichbar” (vgl. auch mit Beispiel im Abschnitt 3.1).

- *Referentielle Bedingungen:* Sie spezifizieren semantische Verbindungen zwischen Paaren von Objektmengen, die über mehrere Komponentensysteme verteilbar sind.

  Beispiel 3.4: In einer Unternehmung gibt es Projektdatenbanksysteme und auch eine eigenständige Personaldatenbank. Man kann eine Bedingung formulieren, die das Vorhandensein der Projektmitarbeiter in der Personaldatenbank fordert.

Alle hier aufgeführten Konsistenzbedingungen sind zunächst statische Bedingungen. Aggregatbedingungen können aber auch transitional sein, z.B. wenn eine Summe nur wachsen darf. Die folgende transitionale oder dynamische Bedingung entsteht für ein föderiertes System aufgrund der mit der Datenverteilung möglichen Datenmigration:

- Beschreibung der erlaubten Migrationen von Daten in föderierten Systemen.

Beispiel 3.5: Produktdaten für ein konkretes Produkt vollziehen logistische Bewegungen durch Migrationen der zutreffenden Daten. Im Unterschied zum Beispiel für systemübergreifende Schlüsselbedingungen werden hier die Bewegungen der einzelnen Datenobjekte und nicht Eigenschaften der Objektmenge betrachtet. Eine solche Bedingung wird jedoch bei der Transparenz des föderierten Schemas als eine Bedingung formuliert sein, die sich bei Verwendung eines objektierten föderierten Schemas auf die Reihenfolge der Existenz bestimmter Objekte bezieht. Es kann sich dabei natürlich auch um eine Objektmigration auf föderierter Ebene handeln.

Es gibt auch Konsistenzbedingungen, deren Überprüfung immer auf eine Datenbank beschränkt sein sollte, d.h. sie sind nur als lokale Bedingungen sinnvoll:

- Domain- oder Attributbedingungen schränken die Werte ein, die ein Attribut annehmen kann. Diese Bedingung ist lokal begrenzt, da ein Attribut selbst nicht auf mehrere Datenbanksysteme verteilt sein kann. Ein komplexes Objekt kann aber als Attribut eine Datenstruktur haben, die Verweise auf in unterschiedlichen Datenbanksystemen befindlichen Objekte enthalten.

- Spezifikation von Zusammenhängen zwischen verschiedenen Attributen eines Objektes durch einen Vergleich ihrer Werte, wenn die Verteilung von Objekten über verschiedene Systeme nicht zugelassen ist.

Es gibt jedoch den Ansatz, semantisch identische Objekte, die in unterschiedlichen Systemen existieren, im föderierten System als ein einziges Objekt zu betrachten. Solche Objekte heißen *Proxy-Objekte*. Außerdem können Objekte, die in verschiedenen Komponentensystemen existieren, auf föderierter Ebene zu einem Aggregat zusammengefaßt werden. In diesen beiden Fällen können verschiedene Attribute eines Objektes auf mehreren Datenbanken liegen. Die letztgenannte Konsistenzbedingung kann dann zu einer datenbankübergreifenden Konsistenzbedingung werden.

4 Mechanismen zur Konsistenzsicherung in FDBSen

Im vorangegangenen Kapitel wurden Konsistenzbedingungen in föderierten Datenbanken behandelt. Dieses Kapitel beschäftigt sich nun mit Mechanismen zur Überwachung globaler Konsistenzbedingungen in föderierten Datenbanksystemen. Soll die Definition von globalen Konsistenzbedingungen in einem föderierten Datenbanksystem auf föderierter Ebene ermöglicht werden, ist zu überlegen, welche Mechanismen die Einhaltung dieser Konsistenzbedingungen sicherstellen sollen. Konsistenzbedingungen auf föderierter Ebene werden nachfolgend auch als *globale Konsistenzbedingungen* bezeichnet. Die Definition solcher Konsistenzbedingungen soll dabei transparent erfolgen, d.h. es sollen die von zentralisierten Datenbanksystemen bekannten Konsistenzbedingungen auch auf föderierter Ebene unterstützt werden. Im Abschnitt 3.3 wurden diese bekannten Konsistenzbedingungen daraufhin untersucht, ob sie innerhalb eines föderierten Datenbanksystems auf mehrere Komponentendatenbanksysteme gleichzeitig wirken können. Die Mechanismen zur Konsistenzsicherung in föderierten Datenbanksystemen sollen dabei lokale, d.h. in Komponentensystemen vorhandene, Mechanismen nutzen. Es gibt in einem föderierten Datenbanksystem also global definierte und auch (eventuell schon vor der Föderierung definierte) lokale Konsistenzbedingungen (siehe. dazu Abb. 4.1).

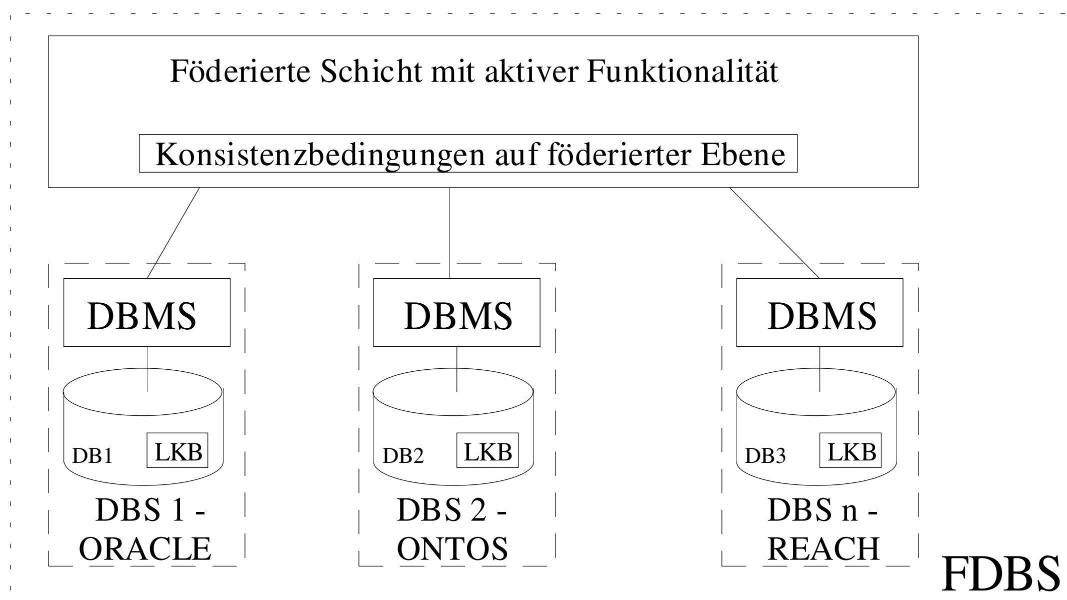

Abbildung 4.1: Lokale und globale Konsistenzbedingungen in einem FDBS

In dieser Abbildung ist zu erkennen, daß die lokalen Konsistenzbedingungen jeweils genau einem Komponentendatenbanksystem zugeordnet sind, während die globalen Konsistenzbedingungen der föderierten Ebene zugeordnet sind. Diese globalen

Konsistenzbedingungen können Daten in mehreren Komponentendatenbanksystemen betreffen. Es werden nun systemübergreifende globale Konsistenzbedingungen daraufhin untersucht, mit welchen lokal vorhandenen und darüber hinaus systemübergreifenden Mechanismen sie in föderierten Systemen überwacht werden können. Um lokale Konsistenzmechanismen nutzen zu können, müssen aus global definierten Konsistenzbedingungen lokale Konsistenzbedingungen abgeleitet werden. Zur Koordinierung der Überwachung der aus global definierten Konsistenzbedingungen, wie z.B. Schlüsselbedingungen oder referenziellen Bedingungen auf föderierter Ebene, abgeleiteten lokalen Konsistenzbedingungen sollen ECA-Regeln genutzt werden. Im Abschnitt 4.2 wird dazu das Konzept der *systemübergreifenden* ECA-Regeln beschrieben und im Abschnitt 4.3 werden die *Systemkomponenten* zur Verarbeitung solcher ECA-Regeln diskutiert. Ein Algorithmus zur Abbildung global definierter ECA-Regeln auf Regeln in Komponentendatenbanksystemen wird im Abschnitt 4.4 entwickelt, wobei im Abschnitt 4.5 an diesem Algorithmus Ergänzungen vorgenommen werden.

4.1 Realisierung global definierter Konsistenzbedingungen mittels lokal vorhandener Konsistenzsicherungsmechanismen

Die Verwendung lokal bereits vorhandener Konsistenzsicherungsmechanismen bedeutet, daß lokal neue Konsistenzbedingungen definiert werden müssen. Dies ist durch die Verwendung der von den Komponentendatenbanksystemen angebotenen deklarativen Konsistenzbedingungen möglich. Die nachträgliche Definition deklarativer Konsistenzbedingungen erfordert die Unterstützung von Schemaevolutionen durch das Komponentendatenbanksystem. Wenn ein solches System keine Schemaevolution unterstützt, aber aktive Fähigkeiten besitzt, müssen stattdessen neue Regeln oder Trigger zur Überwachung abgeleiteter lokaler Konsistenzbedingungen definiert werden. Erlaubt ein Komponentensystem keine Schemaevolution und besitzt es keine aktiven Fähigkeiten, so sollten zur Ermöglichung einer Überwachung abgeleiteter lokaler Konsistenzbedingungen diese aktiven Fähigkeiten z.B. durch den Einsatz einer Zusatzschicht ergänzt werden. Soweit möglich, ist aus Gründen der Effizienz die Verwendung deklarativer Konsistenzbedingungen vorzuziehen.

Im folgenden werden die bekannten Konsistenzbedingungen einzeln darauf untersucht, mit welchen Konsistenzmechanismen sie auf die Komponentensysteme abbildbar sind. Die Definition von Konsistenzbedingungen auf föderierter Ebene, die nur Daten in jeweils einem Komponentensystem betreffen, sollte jedoch vermieden werden. Diese Konsistenzbedingungen können effizienter durch lokale Mechanismen überwacht werden. Was geschieht aber mit den folgenden Bedingungen, wenn sie nachträglich eingefügt werden sollen, und noch nicht vorhanden sind?

4.1.1 Domain- bzw. Attributbedingungen

Domain- oder Attributbedingungen werden in ORACLE durch `CHECK`[4]-Klauseln oder die Angabe der `NOT NULL`-Bedingung direkt im Schema realisiert. Sie können in relationalen Datenbanksystemen, wenn eine Schemamodifikation nicht erlaubt ist, mit einem Trigger (siehe dazu auch Abschnitt 3.1.1) nachgebildet werden. Auf aktiven Komponentendatenbanksystemen kann eine ECA-Regel zur Überwachung einer Domain- oder Attributbedingung benutzt werden. Eine solche Regel muß dann folgendermaßen aufgebaut sein:

- Das hier zu erkennende Ereignis ist die disjunktive Verknüpfung von Einfügen eines neuen Objektes und Verändern des zu überwachenden Attributes.
- Die Bedingung beschreibt die zulässigen Attributzustände.
- Die Aktion beschreibt die Reaktion auf die Verletzung der Bedingung, z.B. die Zurückweisung der Operation.

Die Gestaltung der Aktion hängt davon ab, wie auf eine Konsistenzverletzung reagiert werden soll. So kann vor der Ausführung einer potentiell konsistenzverletzenden Operation geprüft werden, ob durch die Ausführung der Operation eine Konsistenzverletzung auftritt. Dann wird die ECA-Regel als `before` definiert und die Aktion dient zur Anzeige der versuchten Konsistenzverletzung und eventuellen kompensierenden Ersatzanweisungen. Im Gegensatz dazu kann auch eine optimistische Form der Konsistenzsicherung verfolgt werden. Dabei werden die potentiell konsistenzverletzenden Operationen zunächst ausgeführt und wenn sie Konsistenzverletzungen verursacht haben, müssen die Änderungen dieser Operationen rückgängig gemacht oder kompensiert werden.

4.1.2 Konsistenzbedingungen auf Objektebene

Konsistenzbedingungen auf Objektebene (intraobjektive Konsistenzbedingungen) können, wenn keine Proxy-Objekte oder Aggregate betroffen sind, in relationalen Datenbanksystemen durch Trigger überwacht werden, in aktiven Systemen können ECA-Regeln definiert werden. Sie werden wie die o.g. Regel für die Attributbedingungen gestaltet, nur beschreibt die Bedingung der ECA-Regel die Beziehung zwischen den Attributen des Objektes. Beziehungen zwischen Attributen in Aggregaten auf föderierter Ebene sind dagegen anders zu behandeln. Eine systemübergreifende ECA-Regel ist hier nötig, wenn die in Beziehung stehenden

[4]Hinweis: In diesem Abschnitt werden, soweit erforderlich, die Mechanismen des relationalen Datenbanksystems ORACLE verwendet.

Attribute zu Teilobjekten des Aggregates auf verschiedenen Datenbanken gehören. Diese Regel muß Erzeugung und Manipulation von Teilobjekten erkennen. Da diese Teilobjekte auch von lokal arbeitenden Anwendungen verändert werden können, sind lokal wie auch global ausgelöste Datenmanipulationen zu erkennen. Es muß jedoch ein Sonderfall betrachtet werden: Wird ein Teilobjekt lokal erzeugt und das andere Teilobjekt, welches das zur Bedingung gehörige zweite Attribut enthält, existiert noch nicht, sind verschiedene Möglichkeiten der Reaktion auf die damit möglicherweise entstandene Konsistenzverletzung möglich. Es kann zum einen ein Teilobjekt auf der betreffenden Datenbank mit einem die Konsistenzbedingung erfüllenden Standardwert für das betroffene Attribut erzeugt werden. Die Konsistenzbedingung kann auch so formuliert werden, daß sie bei einem fehlenden Teilobjekt nicht verletzt ist. Die zugehörige ECA-Regel zur Überwachung von Beziehungen zwischen Attributen innerhalb von Aggregaten muß folgendermaßen wirken:

- Das Ereignis ist ein komplexes Ereignis aus Operationen (Erstellung und Modifikation der betreffenden Attribute) auf den Teilobjekten.
- Die Bedingung der Regel ist eine systemübergreifende Bedingung zwischen den betreffenden Attributen gemäß der Konsistenzbedingung
- Die Aktion zeigt die Konsistenzverletzung an oder kompensiert die entstandene Konsistenzverletzung.

Außerdem kann es in einem föderierten Datenbanksystem Proxy-Objekte geben. Hier gibt es zwei semantisch gleichwertige Objekte auf verschiedenen Komponentensystemen, die ein- und dasselbe Objekt der Realität repräsentieren und auf föderierter Ebene als ein Objekt betrachtet werden. In diesem Fall müssen Veränderungen an diesen Proxy-Objekten auf beiden Komponentensystemen überwacht werden. Erfolgt eine Konsistenzverletzung durch eine Operation auf föderierter Ebene, so sind beide Proxy-Objekte von dieser Veränderung betroffen und eine Konsistenzverletzung kann wie bei normalen Objekten behandelt werden. Wird durch eine lokale Operation eines der beiden Proxy-Objekte inkonsistent, so könnte mittels einer ECA-Regel auf föderierter Ebene diese Konsistenzverletzung durch Wiedereintragen der konsistenten Attributwerte des anderen Proxy-Objektes kompensiert werden. Für jedes der beiden Proxy-Objekte ist eine ECA-Regel nötig, die folgendermaßen formuliert sein kann:

- Das Ereignis zeigt eine Operation an einem für die Konsistenzbedingung relevanten Attribut eines Proxy-Objektes an
- Die Bedingung beschreibt die Beziehung zwischen zwei Attributen

- Die Aktion ist hier die Kompensation durch die Werte des anderen zugehörigen Proxy-Objektes, die innerhalb der Aktion durch eine Datenbankanfrage zu ermitteln sind.

Beispiel 4.1: In zwei Komponentendatenbanken `DB1` und `DB2` sind Informationen zu Personen gespeichert. Dazu gehören neben `Namen`, `Vornamen` und `Geburtsdatum` auch das Attribut `Jahr der Einschulung`. Es gilt die Konsistenzbedingung, daß das Geburtsjahr und das Einschulungsjahr eine Differenz von mindestens 6 Jahren zueinander aufweisen müssen. In beiden Datenbanken ist nun ein und dieselbe Person mit dem Namen Müller, dem Vornamen Bert, dem Geburtsjahr 1972 und dem Einschulungsjahr 1978 gespeichert. Auf der Datenbank `DB1` wird nun als Einschulungsjahr 1976 eingetragen, womit die Konsistenzbedingung verletzt ist. Zur Bereinigung der Inkonsistenz auf der Datenbank DB1 kann der Wert „1978” aus der Datenbank DB2 für das Attribut Einschulungsjahr eingetragen werden.

4.1.3 Globale Schlüsselbedingung

Der Mechanismus zur Überprüfung einer globalen Schlüsselbedingung ergibt sich aus der Semantik des Schlüsselbegriffes. Zunächst bedeutet eine globale Schlüsselbedingung automatisch, daß das Attribut, welches globales Schlüsselattribut ist, auch auf lokaler Ebene die Schlüsseleigenschaft erfüllen muß. Diese lokal begrenzten Schlüsseleigenschaften lassen sich durch bereits im Komponentensystem existierende Mechanismen, wie die `UNIQUE`-Bedingung in ORACLE oder ONTOS, realisieren. Bei Verwendung dieser deklarativen Konstrukte muß eine Schemamanipulation zur Systemlaufzeit möglich sein. Aus Gründen der Effizienz ist hier eine Schemamanipulation zu favorisieren. Es kann jedoch der Fall eintreten, daß die vorhandenen Daten diese lokale Schlüsselbedingung nicht erfüllen würden. In diesem Fall müßten zunächst die vorhandenen Daten so geändert werden, daß sie die lokale Schlüsselbedingung erfüllen, da ansonsten die Schemaänderung vom lokalen Datenbankmanagementsystem zurückgewiesen wird. In aktiven Komponentendatenbanksystemen ohne Unterstützung einer Schemaevolution kann eine ECA-Regel die Einhaltung einer lokalen Schlüsselbedingung gewährleisten. Diese Regel überwacht die Erstellung eines neuen Objektes oder eine Manipulation von Schlüsselattributen bestehender Objekte. In einem relationalen Komponentendatenbanksystem ohne Unterstützung der Schemaevolution kann ein Trigger diese Aufgabe übernehmen. Hier wird noch einmal deutlich, daß Komponentensysteme eines föderierten Datenbanksystems, in dem Konsistenzsicherung unter Verwendung der in dieser Arbeit beschriebenen Architektur betrieben werden soll, entweder Schemaevolution unterstützen oder aktive Mechanismen besitzen müssen. Sollte ein Komponentensystem keine dieser beiden Voraussetzungen erfüllen, kann eine Schicht zur Bereitstellung aktiver Funktionalität dem System beigefügt werden. Bei

der weiteren Behandlung globaler Konsistenzbedingungen in diesem Abschnitt wird deshalb davon ausgegangen, daß die Komponentendatenbanksysteme diese Voraussetzungen erfüllen.

Die Betrachtung der globalen Schlüsselbedingung zeigt außerdem deutlich, daß bei der Einführung solcher global gültigen Konsistenzbedingungen Abstimmungen zwischen den globalen und den lokalen Datenbankadministratoren notwendig sind. Zur Sicherung der Einhaltung einer globalen Schlüsselbedingung bei Dateneinfügung und Veränderung auf globaler Ebene kann eine systemübergreifende ECA-Regel, wie sie im folgenden Abschnitt 4.2 beschrieben wird, dienen. Diese systemübergreifende ECA-Regel wirkt dann folgendermaßen:

- Ereignisse können Manipulationsoperationen an Schlüsselattributen und Erzeugungsoperationen für neue Objekte sowohl auf lokaler als auch auf globaler Ebene sein.
- Die Bedingung der ECA-Regel beschreibt den Sachverhalt, daß der neue Wert für das betreffende Attribut schon in einer der anderen Datenbanken vorhanden ist.
- Die Aktion ist hier die Zurückweisung der Operation.

Die hier betrachtete globale ECA-Regel hat (wie auch die bereits beschriebenen ECA-Regeln) ein systemübergreifendes Ereignis und eine mehrere Komponentensysteme betreffende Bedingung. Die Verarbeitung dieser Regel muß dann folgendermaßen erfolgen: Nach dem Eintreten eines Ereignisses, das eine global oder lokal ausgeführte Operation an einem Schlüsselattribut oder die Erstellung eines neuen Objektes anzeigt, muß diese Operation einer globalen Regelverarbeitungskomponente angezeigt werden. Daraufhin wird eine systemübergreifende Datenbankanfrage nach dem neuen Schlüsselwert durchgeführt. Bei noch nicht erfolgter Operation ist der Wert des Schlüsselattributes in keiner Datenbank vorhanden. Wurde eine Verletzung der globalen Schlüsselbedingung erkannt, kann die Aktion der ECA-Regel die Operation zurückweisen.

Außerdem ist es in einigen Fällen unumgänglich, lokal bestehende Schlüssel zu globalen Schlüsseln zu erweitern. Dies bedeutet, daß im globalen Datenmodell zu bereits lokal als Schlüsselattributen definierten Teilschlüsseln auf föderierter Ebene neue Teilschlüssel hinzukommen. Diese Schlüsselerweiterung ist z.B. dann erforderlich, wenn semantische Heterogenität auftritt. So können in zwei verschiedenen Personaldatenbanken Personen unter dem gleichen Schlüssel gespeichert sein, obwohl diese Personen nicht miteinander identisch sind. Ein solcher Fall würde eine vorhandene globale Schlüsselbedingung verletzen. Es muß nun ein Attribut gefunden werden, das den lokalen Schlüssel zu einem global gültigen Schlüssel erweitert. Dazu können sich vorhandene Attribute eignen, es kann auch ein neues Attribut hinzugefügt werden, welches dann automatisch generierte Werte erhalten kann. Eignen sich vorhandene Attribute zur Schlüsselerweiterung, kann die nun erweiterte Schlüsselbedingung

lokal auf ihre Einhaltung überprüft werden. Ein hinzugefügtes Attribut könnte durch Änderung der lokalen Datenstruktur abgebildet werden, d.h. im lokalen Schema wird ein Attribut hinzugefügt. Dies erfordert jedoch wieder die Unterstützung einer Schemaevolution durch das betreffende Komponentendatenbanksystem. Können die erforderlichen Änderungen nicht zur Systemlaufzeit erfolgen, muß dieses System eventuell seine Arbeit für einen kurzen Zeitraum unterbrechen, so daß die erforderlichen Manipulationen ermöglicht werden. Abschließend wird ein Beispiel für eine globale Schlüsselbedingung behandelt.

Beispiel 4.2: Im Abschnitt 2.3 wurden exemplarisch Schemata für die Komponentensysteme sowie ein Teil eines föderierten Schemas aufgebaut. Dieses föderierte Schema enthält eine Klasse `Produkt`, für die ein Schlüssel in der Form

```
key (seriennummer,typenbezeichnung);
```

vereinbart wurde. Zu dieser Klasse wurden Tupel der lokalen Tabellen `Exemplar` und `Lagerung` zusammengefaßt. Somit enthält die Extension der Klasse Produkt Objekte, die verschiedenen Datenbanken zuzuordnen sind. Da vor der Integration bereits lokal Schlüssel durch

```
PRIMARY KEY (SerienNr, Typ)
```

vereinbart waren, gibt es keine Probleme bei der Überwachung lokaler Schlüssel. Durch die lokalen Schlüsselbedingungen wird die Einhaltung dieser globalen Schlüsselbedingung bereits gewährleistet. Eine zusätzliche globale ECA-Regel ist nur dann nötig, wenn in verschiedenen Datenbanken Objekte mit gleicher Semantik redundanzfrei gespeichert werden sollen.

4.1.4 Globale Aggregatbedingungen

Die Überwachung globaler Aggregatbedingungen wird ähnlich wie die globale Schlüsselbedingung im vorangegangen Abschnitt realisiert. Die Ableitung lokaler Bedingungen ist dazu auch hier nötig. Das Prinzip des Zusammenwirkens zwischen lokalen Bedingungen und globalen Konsistenzsicherungsmechanismen wird am Beispiel der Summe aller Attributwerte in Beziehung zu einem Grenzwert beschrieben. Es wird nun angenommen, daß mehrere Datenbanken Objekte, für welche die Konsistenzbedingung gelten soll, enthalten. Für jede dieser Komponentendatenbanken werden lokal die Summen berechnet. Hierfür werden erneut aktive Mechanismen der Komponentendatenbanksysteme, wie z.B. Trigger oder ECA-Regeln gebraucht. Eine globale ECA-Regel kontrolliert Veränderungen an den lokal berechneten Summen. Wird die Gesamtsumme, die im Bedingungsteil der globalen ECA-Regel ermittelt werden kann, zu groß, so muß eine Reaktion erfolgen. Bei der Wahl dieses Ansatzes ist zu beachten, daß die ursprünglich konsistenzverletzenden Datenbankoperationen nicht

überwacht werden, weshalb eine Zurücksetzung dieser Operationen erschwert wird. Es ist dann zu überlegen, wie solche Fehler kompensiert werden können.

4.1.5 Rekursive Bedingungen

Auch die rekursiven Bedingungen sind mit der gleichen Idee zu überwachen. Lokale Mechanismen melden, wie beschrieben, die Verletzung der rekursiven Bedingung auf globaler Ebene, eine globale Regel prüft, ob bei Verletzung einer abgeleiteten lokalen Bedingung die Konsistenzbedingung global erfüllt sein könnte. Ein Beispiel soll dies verdeutlichen:

> **Beispiel 4.3:** Die Reiseverbindungsdatenbank aus Kapitel 3.3 enthält eine global definierte rekursive Konsistenzbedingung. Diese Bedingung besagt, daß jeder Ort innerhalb der globalen Datenbank von jedem anderen aus erreichbar sein soll. Für jede lokale Datenbank kann man diese Bedingung zunächst lokal fordern, was bedeutet, daß innerhalb der Datenbank alle Orte untereinander erreichbar sind. Bei der Vereinigung der Daten auf globaler Ebene muß dann jeweils eine Reiseverbindung existieren, die zwei Orte aus verschiedenen Datenbanken verbindet, wenn die rekursive Konsistenzbedingung global erfüllt sein soll. Dieser globale Test kann mit einer ECA-Regel erfolgen. Die die erwähnte Regel auslösenden Ereignisse sind lokal und global durchgeführte Hinzufügungen neuer Verbindungen. Als Bedingung der ECA-Regel bietet sich eine Query an, die untersucht, ob der zweite zur Verbindung gehörende Ort in einer der anderen Datenbanken bereits existiert und damit auch erreichbar ist. Eine weitere, ähnlich aufgebaute ECA-Regel muß dann noch die Löschung von Verbindungen überwachen.

4.1.6 Referentielle Bedingungen

Bei den systemübergreifenden referentiellen Bedingungen wurde zwischen zwei Ausprägungen unterschieden. Zunächst wird der Fall betrachtet, daß zwischen zwei auf zwei verschiedenen Datenbanken befindlichen Objektmengen eine referentielle Beziehung besteht. Ein referenzierendes Objekt muß beim Aufbau der Referenz überwacht werden. Es ist zu ermitteln, ob das referenzierte Objekt existiert. So muß, wenn Tupel auf einem relationalen Datenbanksystem referenziert werden sollen, dort durch eine Anfrage die Existenz des referenzierten Tupels nachgewiesen werden. Diese Aufgabe übernimmt auch hier eine globale ECA-Regel. Diese ECA-Regel könnte wie folgt aufgebaut sein:

- Das Ereignis zeigt einen Referenzaufbau an.
- Die Bedingung der Regel besagt, daß das referenzierte Objekt nicht existiert.
- Mittels der Aktion kann die Konsistenzverletzung angezeigt oder verhindert werden.

Das Gegenstück bildet eine globale ECA-Regel für referenzierte Objekte. Sie überwacht die Modifikation oder Löschung referenzierter Objekte oder Tupel. Diese Regel hat die folgenden Komponenten:

- Das Ereignis zeigt die Modifikation oder Löschung eines referenzierten Objektes an.
- Die Bedingung besagt, daß mindestens ein referenzierendes Objektes existiert.
- Mittels der Aktion kann auch hier die Konsistenzverletzung angezeigt oder verhindert werden.

Der zweite Fall einer systemübergreifenden referentiellen Konsistenzbedingung ist etwas komplizierter. Hier sind die Objektmengen selbst noch über Datenbanken verteilt. Es müssen auch hier Regeln zur Überwachung sowohl referenzierter als auch referenzierender Objekte oder Tupel bereitgestellt werden. Bei der Regelübersetzung ergeben sich für die Manipulationen an den referenzierenden oder referenzierten Objekten jeweils verschiedene Operationen auf den unterschiedlichen Datenbanksystemen.

Auf jeder Datenbank, die im Zusammenhang mit einer systemübergreifenden referenziellen Bedingung referenzierte oder referenzierende Objekte oder Tupel enthält, müssen Regeln zur Überwachung von Manipulationen an diesen Objekten eingeführt werden. Diese Regeln bzw. Tupel müssen jedoch bei der Übersetzung der global definierten ECA-Regel aus dieser durch einen geeigneten Mechanismus abgeleitet werden. Einen Algorithmus für diesen Ableitungsprozeß wird daher im Abschnitt 4.4 dieser Arbeit behandelt. Im jetzt folgenden nächsten Abschnitt 4.2 werden die in diesem Abschnitt bereits häufiger erwähnten globalen ECA-Regeln für föderierte Systeme behandelt.

4.2 ECA-Regeln in föderierten Systemen

In diesem Kapitel soll untersucht werden, wie ECA-Regeln in föderierten Systemen unterstützt werden können. Diese global gültigen ECA-Regeln sollen, ähnlich wie in zentralisierten Datenbanksystemen, auch zur Konsistenzsicherung in föderierten Systemen verwendet werden. Die Idee der systemübergreifenden ECA-Regeln entstand in Anlehnung an die verteilte Datenhaltung und die systemübergreifenden Datenbankanfragen. Ein Ansatz zur Realisierung des ECA-Regel Konzeptes in föderierten Umgebungen wird von [TC95] vorgeschlagen. Hierbei wird der Standpunkt vertreten, daß eventuell schon bestehende aktive Funktionalität der Komponentendatenbanksysteme verwendet werden soll.

Die systemübergreifende Wirkungsweise der ECA-Regeln kann in zwei unterschiedlichen Sichtweisen betrachtet werden. Zunächst kann eine ECA-Regel systemübergreifend sein, indem sich die Bestandteile jeweils als Ganzes auf verschiedene Datenbanksysteme beziehen. Aber auch die Bestandteile selbst können sich auf mehrere verschiedene Datenbanksysteme oder Datenbanken beziehen.

Ein komplexes Ereignis einer ECA-Regel kann aus mehreren primitiven Ereignissen zusammengesetzt werden, die auf verschiedenen lokalen Datenbanken erkannt werden müssen. Die Bedingung einer systemübergreifenden ECA-Regel könnte systemübergreifende Datenbankanfrage sein, während die Aktion z.B. Konsistenzwiederherstellungsoperationen auf mehreren Komponentendatenbanksystemen durchführen soll. Zunächst wird die Notation für ECA-Regeln mit in sich systemübergreifenden Bestandteilen vorgestellt, die auch im Verlauf dieser Arbeit verwendet wird:

```
RULE <Regelname>
EVENT       e1 eop1 e2 eop2...eopn-1 en
CONDITION   c2 cop1 e2 cop2...copn-1 em
ACTION      a1;a2;...;ak
```

Das Schlüsselwort `RULE` steht am Anfang einer jeden Regel, danach folgt der Bezeichner zur Identifikation der Regel. Mit `EVENT` wird der Ereignisteil einer Regel deklariert. `CONDITION` zeigt an, daß nun der Bedingungsteil der Regel folgt, während `ACTION` die Aktionskomponente einleitet. Wenn ein Ereignis oder eine Bedingung aus mehreren Teilen besteht, werden diese Teile mit den Operatoren `eop` bzw. `cop` verknüpft. Auf diese Operatoren wird in den nächsten Abschnitten ausführlicher eingegangen.

Es wird hier darauf hingewiesen, daß eine Zuordnung von Teilen der Regelkomponenten zu den Datenbanken oder Datenbanksystemen bei der Spezifikation einer ECA-Regel nicht explizit erfolgen muß, da diese Zuordnung im Data Dictionary des föderierten Schemas beschrieben ist. Bei der Integration der Exportschemata der Komponentendatenbanksysteme

zu hier nur einem föderierten Schema werden diese Informationen im Data Dictionary des föderierten Systems abgelegt. So sieht ein Anwendungsprogrammierer nur die Objekte des föderierten Schemas nicht jedoch ihre Entsprechungen für die unterliegenden Komponentensysteme. Im Verlauf der Regelübersetzung werden diese Informationen durch den Zugriff auf die Metadaten im föderierten Schema aufgelöst.

Ausgehend vom Szenario einer eng gekoppelten Föderation im Abschnitt 2.3 wird jetzt eine der dort aufgestellten Konsistenzbedingungen als global definierte ECA-Regel mit der gerade eingeführten Notation für ECA-Regeln formuliert.

Beispiel 4.4: Das Lagereingangsdatum in der Lagerdatenbank darf nicht früher als das Produktionsdatum sein.

```
RULE   Eingangsdatum
EVENT       lagerung::setEingDatum(datum edatum)
CONDITION   exemplar.produktionsdatum >= edatum
ACTION      <Wiederherstellung der Konsistenz>
```

Diese Regel enthält eine systemübergreifende Bedingung. Das Objekt `exemplar` entspricht einem Tupel der Relation `Exemplar` in der relationalen Produktionsdatenbank, während `lagerung` einem Tupel der Relation `Lagerung` der ebenfalls relationalen Lagerdatenbank entspricht. Die Verknüpfung der Bedingungsteile erfolgt mit einem Operator, der für die beteiligten Datentypen gültig sein muß. Dieser Operator ist hier ein Vergleichsoperator für Datumsangaben.

4.2.1 Ereignisse in föderierten Umgebungen

Ereignisse auf föderierter Ebene können systemübergreifend sein. Dies ist der Fall, wenn ein komplexes Ereignis Teilereignisse enthält, die Operationen auf verschiedenen Komponentensystemen anzeigen. Es gibt eine Reihe von Ereignissen, die direkt im föderierten Datenbanksystem erkannt werden sollten. Dazu gehören alle Ereignisse, die keine Datenbankoperationen auf Daten in Komponentendatenbanken anzeigen, wie z.B. Zeitereignisse oder nur die globale Datenbank betreffende Ereignisse.

Die Teilereignisse werden mit den in der Notation für die ECA-Regeln als $eop_1 ... eop_n$ bezeichneten Operatoren zu komplexen, systemübergreifenden Ereignissen verknüpft. Diese Ereignisoperatoren können die in der Tabelle 3 im Abschnitt 3.2 beschriebenen Operatoren sein.

4.2.2 Systemübergreifende Bedingungen von ECA-Regeln

Die Bedingung einer ECA-Regel kann nach [DBB+88, DHW95] ein Prädikat oder eine Datenbankanfrage sein. Eine Bedingung als Prädikat kann systemübergreifend sein, wenn das Prädikat eine z.B. logische Verknüpfung mehrerer Prädikate von in verschiedenen Datenbanken befindlichen Objekten ist. Ein Vergleich zwischen Attributen von Objekten oder Tupeln, die in verschiedenen Datenbanken möglicherweise unterschiedlicher Datenbanksysteme gespeichert sind, ist eine weitere Form für eine systemübergreifende Bedingung einer ECA-Regel. Die Bedingung einer ECA-Regel kann auch eine systemübergreifende Datenbankanfrage sein, die mit einem geeigneten Anfragewerkzeug durchzuführen ist.

Wie die ECA-Regel aus Abschnitt 4.2 für eine Konsistenzbedingung zeigt, treten auch Vergleiche zwischen Attributen als Bedingung über mehreren Systemen auf. Bei der Auswertung dieser Bedingung muß die Semantik des Vergleiches, hier ein Vergleich zweier Datumsangaben, berücksichtigt werden. Am Bedingungsteil dieser Regel wird auch deutlich, daß ein Parameter der auslösenden Methode zur Bedingungsevaluierung herangezogen werden kann. Wenn man davon ausgeht, daß ein solcher Parameter einer Methode, die zu einer Datenmanipulation dient, Attributwert eines Objektes oder Tupels werden soll, muß man testen können, ob dieser (potentielle) Attributwert eine Konsistenzbedingung verletzt oder nicht.

4.2.3 Systemübergreifende Aktionen

Systemübergreifenden Aktionen sind Aktionen, die über mehrerer Datenbanksysteme hinweg ausgeführt werden sollen, aber dabei insgesamt jedoch als *eine* Einheit, ähnlich wie bei einer Transaktion, zu betrachten und zu behandeln sind. Systemübergreifende Aktionen werden gebraucht, um im Falle einer Konsistenzverletzung konsistenzwiederherstellende Maßnahmen auf mehreren Datenbanksystemen einleiten zu können.

Diese mehrere Komponentensysteme betreffenden Aktionen sind nicht identisch mit mehreren getrennten Regeln mit gleichem Ereignis- und Bedingungsteil, aber verschiedenen Aktionsteil und deshalb *nicht* durch solche Regeln zu realisieren. Der Unterschied zwischen einer ECA-Regel mit systemübergreifender Aktion und mehreren ECA-Regeln mit gleichem Ereignis- und Bedingungsteil besteht in der Abarbeitung der Regeln. Die Regelverarbeitungskomponente muß über die Reihenfolge der abzuarbeitenden Regel entscheiden. Die Reihenfolge ist z.T. durch Prioritätenvorgabe zu beeinflussen. Die Betrachtung der Aktion als untrennbare Einheit ist jedoch nicht gegeben, da zwischenzeitlich andere vorhandene Regeln bearbeitet werden können.

Systemübergreifende Aktionen müssen Datenmanipulationen durchführen, wenn eine optimistische Form der Konsistenzsicherung durchgeführt werden soll. Dies bedeutet, daß erst nach Ausführung einer Datenbankoperation kontrolliert wird, ob eine Konsistenzverletzung vorliegt. Werden jedoch vor der Ausführung von Datenbankoperationen diese auf mögliche Konsistenzverletzung hin überprüft, so kann im Falle einer Konsistenzverletzung die Ausführung dieser Operationen verhindert werden. In diesem Fall können die Aktionen lediglich in Hinweisen auf eine (versuchte) Konsistenzverletzung bestehen, die ggf. jedoch auch an mehrere Systeme zu senden sind.

Die Kopplungsmodi der ECA-Regeln stehen in einem engen Zusammenhang zum Transaktionskonzept des unterliegenden Datenbankmanagementsystems, denn sie legen die Beziehung der Bestandteile der ECA-Regeln zueinander fest. In föderierten Datenbanksystemen ist ein oftmals offen geschachteltes Transaktionskonzept [BGS95] zu finden und nicht das aus zentralen Datenbanken bekannte ACID-Transaktionskonzept [Vos94]. Bei Verwendung eines offen geschachtelten Transaktionsmodells müssen dann ggf. innerhalb einer Aktion einer ECA-Regel auf mehreren Datenbanksystemen die Effekte von Transaktionen kompensiert werden. Um jedoch überhaupt Regeln auf einem föderierten Datenbanksystem nutzen zu können, muß zunächst ein Regelverarbeitungssystem geschaffen werden.

4.3 Architektur eines Regelverarbeitungssystems in FBDSen

In einem „aktiven” föderierten Datenbanksystem müssen mehrere, zu den Komponentendatenbanksystemen gehörende Regelverarbeitungskomponenten miteinander kooperieren, um eine auf mehreren Systemen wirkende ECA-Regel abzuarbeiten. In den folgenden Abschnitten wird untersucht, welche Komponenten das sind und wie sie zusammenarbeiten können. Dabei wird davon ausgegangen, daß vorhandene aktive Fähigkeiten der Komponentensysteme verwendet werden sollen, um im Fall bereits aktiver Komponentendatenbanksysteme nicht noch eine andere aktive Funktionalität zu implementieren.

4.3.1 Architekturvorschlag

In föderierten Datenbanksystemen werden die Daten heterogener, autonomer Datenbanksysteme durch eine einheitliche Schnittstelle zugänglich gemacht. Folgerichtig können an aktiven föderierten Datenbanksystemen auch heterogene aktive Datenbanksysteme, die verschiedene aktive Mechanismen besitzen, beteiligt sein. Die zu einer globalen Konsistenzsicherung hier erwünschte Kooperation benötigt zur Überwachung globaler Konsistenzbedingungen die Koordination durch eine globale Komponente.

Die aktiven Systemkomponenten eines aktiven föderierten Datenbanksystems werden im weiteren Verlauf dieser Arbeit als Regelmanager bezeichnet. Es wird von [TC95] eine Unterteilung in

- *globale Regelmanager* und
- *lokale Regelmanager*

vorgeschlagen.

Die lokalen Regelmanager sind dabei die Systemkomponenten zur Erkennung lokaler Ereignisse, während die globalen Regelmanager die Arbeit der lokalen Regelmanager koordinieren. Die Aufgaben dieser Regelmanager werden im Verlauf dieses Abschnittes noch näher betrachtet. Die folgende Abbildung 4.1 gibt zunächst einen Überblick über eine mögliche Architektur eines Regelverarbeitungssystems in einem föderierten Datenbanksystem. Hier ist zu erkennen, daß die Benutzung bereits vorhandener aktiver Fähigkeiten der Komponentendatenbanksysteme als lokale Regelmanager erwünscht ist. In dieser Skizze werden Ansatzweise gleichzeitig die Realisierungsmöglichkeiten von lokalen Regelmanagern aufgezeigt.

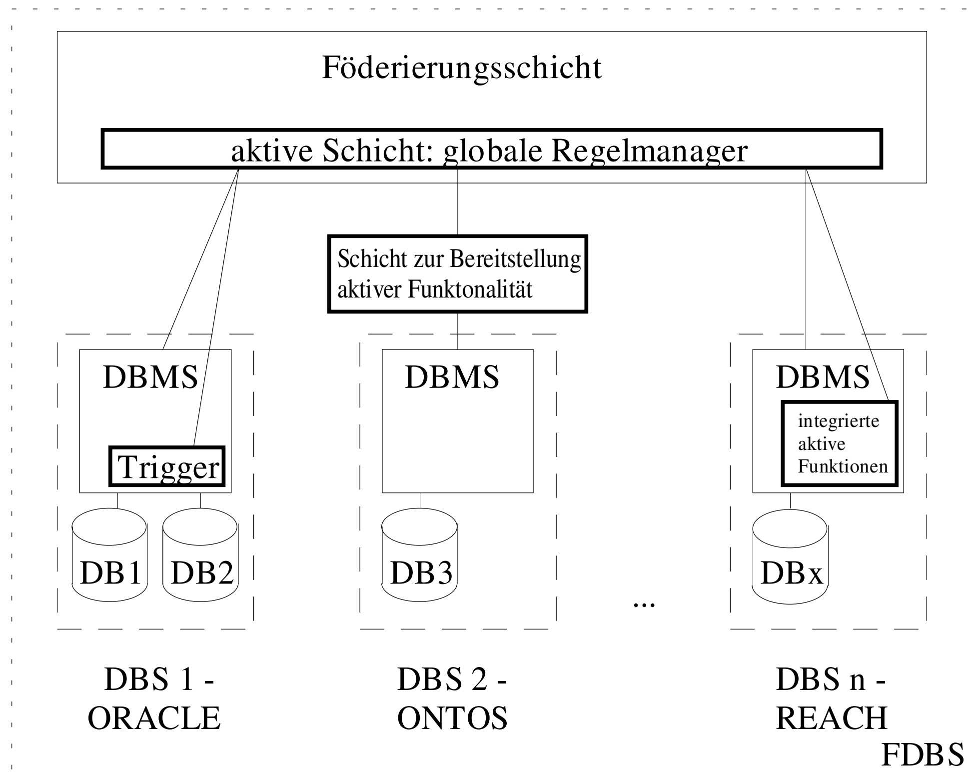

Abbildung 4.2 Architekturvorschlag zur Verarbeitung von ECA-Regeln in FDBSen

Wie hier zu sehen ist, sind die globalen Regelmanager der föderierten Ebene zugeordnet. Sie dienen Koordinationszwecken bei der Abarbeitung der systemübergreifenden ECA-Regeln. Die globalen Regelmanager sollen die folgenden Aufgaben übernehmen:

- Erkennen primitiver Ereignisse auf föderierter Ebene und Zusammensetzen komplexer Ereignisse,
- Auswerten von systemübergreifenden Bedingungen,
- Start systemübergreifender Aktionen mittels geeigneter Transaktionsmechanismen oder Ausführung einfacher Aktionen.

Die lokalen Regelmanager dagegen übernehmen die beiden folgenden Aufgaben:

- Erkennung von lokal eintretenden Ereignissen
- Weitermeldung lokal erkannter Ereignisse zum zuständigen globalen Regelmanager

Die globalen Regelmanager sollen die bereits im Abschnitt 4.2.1 erwähnten, auf die föderierte Ebene begrenzten Ereignisse detektieren. Die lokalen Regelmanager erkennen lokal eintretende Ereignisse und melden sie an die zugehörigen globalen Regelmanager weiter. Die globalen Regelmanager müssen die in einer ECA-Regel spezifizierten komplexen Ereignisse aus global zu erkennenden Ereignissen und den eingehenden Meldungen der lokalen Regelmanager zusammensetzen. Wird das komplexe Ereignis der global definierten ECA-Regel erkannt, kann der globale Regelmanager die Auswertung der Bedingung vornehmen. Im Anschluß daran muß bei erfüllter Bedingung die Aktion ausgeführt werden.

4.3.2 Ereignisdetektion in föderierten Systemen

Wenn ein komplexes Ereignis systemübergreifend ist, besteht es u.a. aus lokal zu detektierenden, primitiven Ereignissen, die durch die Operatoren der Ereignisalgebra miteinander verknüpft sind. Die lokal eintretenden Ereignisse werden lokal detektiert und dem betreffenden globalen Regelmanager signalisiert.

Es muß nun überlegt werden, welche Mechanismen zur Detektion lokal eintretender Ereignisse Verwendung finden sollen. Hier gibt es bezüglich der Ausstattung der Komponentendatenbanksysteme prinzipiell die folgenden Möglichkeiten. Ein Komponentendatenbanksystem unterstützt:

- aktive Regeln,
- Trigger oder

- keine aktiven Fähigkeiten.

Für den Fall, daß das Komponentendatenbanksystem, auf dem lokale Ereignisse zu erkennen sind, bereits aktive Funktionalität besitzt, sollte diese auch benutzt werden. Die Meldung lokal eintretender Ereignisse kann also mittels einer lokal definierten ECA-Regel erfolgen, die beim Eintreten des gewünschten lokalen Ereignisses den zugehörigen globalen Regelmanager benachrichtigen.

Trigger (z.B. in relationalen Datenbanksystemen) könnten diese Aufgabe ebenfalls gut übernehmen. Diese Trigger sind dazu auf ihre Leistungsfähigkeit zu untersuchen. Sie lassen sich im Falle der Eignung dazu verwenden, Datenmanipulationen zu propagieren. Bei der beabsichtigten Verwendung eines Triggers muß beachtet werden, daß durch den `THEN`-Teil (oder PL-SQL Block bei ORACLE-Triggern) des Triggers eine Benachrichtigung des globalen Regelmanagers möglich ist. Es muß also die Möglichkeit eines externen Funktionsaufrufes vorhanden sein, um mittels dieses Funktionsaufrufes einen globalen Regelmanager zu benachrichtigen.

Wenn ein Komponentendatenbanksystem jedoch keine aktive Funktionalität besitzt, oder die vorhandenen aktiven Mechanismen nicht zur Weitermeldung lokal eintretender Ereignisse nutzbar sind, muß eine aktive Zusatzschicht für dieses System eingerichtet werden (siehe [Kot93, CW92]). Diese Zwischenschicht könnte z.B. die Definition und Abarbeitung von ECA-Regeln ermöglichen. Die Ereignisdetektion kann in relationalen wie auch in objektorientierte Systemen durch das aus Abschnitt 3.2 bekannte „Method-Wrapping”, d.h. das „Einpacken” von Datenbankoperationen durch speziellen Programmcode zur Ereignisdetektion erfolgen. Wie bereits erwähnt, müssen dazu auch lokal laufende Anwendungen neu übersetzt werden. Oftmals sind Änderungen in lokalen Anwendungen nicht erwünscht oder aufgrund nicht mehr vorhandener Quelltexte nicht möglich. Dann können nur Operationen, die durch das globale System auf der lokalen Datenbank ausgeführt werden, erkannt werden.

Ein weiteres Problem tritt bei der Verwendung des `THEN`-Operators und aller Operatoren der Ereignisalgebra zu Tage, die Überwachungsintervalle benötigen. Durch unterschiedliche Signallaufzeiten bei der Kommunikation zwischen den Komponentensystemen kann es hier zur Fehldetektion des zusammengesetzten systemübergreifenden Ereignisses kommen. Diese Problematik wird im später folgenden Abschnitt 5.3 näher betrachtet.

4.4 Ableitung lokaler Regeln aus global definierten Regeln

Wie bereits erwähnt, soll die Abarbeitung der global definierten ECA-Regeln durch globale und lokale Regelmanager erfolgen. Dazu ist die Aufstellung lokaler und globaler Regeln nötig, die aus der ursprünglich definierten globalen Regel abgeleitet werden müssen. Die abgeleiteten

lokalen Regeln sind jedoch nur eine Zwischenstufe, wenn im Komponentensystem andere Mechanismen als ECA-Regeln verwendet werden sollen. Die Erstellung dieser *vorläufigen* lokalen Regeln kann automatisch erfolgen. Im Abschnitt 4.4.1 wird ein Algorithmus für diesen Ableitungsprozeß betrachtet.

Daran anschließend werden die Trigger der relationalen Datenbankmanagementsysteme auf ihre Nutzbarkeit zur Weitermeldung lokaler Ereignisse untersucht. Ist diese Fähigkeit gegeben, kann die Übersetzung von lokalen E(C)A-Regeln in einen Befehl zur Generation eines Triggers vorgenommen werden. Eine weitere Schwierigkeit besteht darin, wie die solcherart entstandenen Regeln oder Trigger mit bestehenden lokal definierten Regeln bzw. Triggern harmonieren oder ob Wechselwirkungen entstehen, da sie in eine auch lokal nutzbare Regelbasis integriert werden.

4.4.1 Realisierungsvorschlag für die Ableitung

Im folgenden wird ein Algorithmus grob beschrieben, der eine global definierte ECA-Regeln auf lokale Regeln und eine globale Koordinationsregel automatisch abbildet. Er besteht im wesentlichen aus drei Teilen:

1. Übersetzung der lokalen Ereignisse mittels Metainformationen und Zuordnung zu ihrem Komponentendatenbanksystem

2. Ableitung lokaler Regeln zur Ereigniserkennung und Aufstellung einer globalen Koordinationsregel aus einer (durch den Regelprogrammierer) global definierten Regel,

3. Anpassung der abgeleiteten lokalen Regeln an die in Komponentendatenbanksysteme vorhandenen aktiven Mechanismen.

In föderierten Datenbanksystemen werden Metainformationen verwaltet, die die Zuordnung von Objektspezifikationen, Methoden etc. zu den entsprechenden Objekten (oder Tupeln in relationalen Datenbanksystemen) und den lokalen Methoden (oder SQL-Operationen in relationalen Datenbanksystemen) beschreiben. In föderierten Systemen ist es möglich, daß verschiedene Datenbanksysteme semantisch gleichwertige Daten enthalten und der Zugriff auf föderierter Ebene dann einheitlich erfolgt. Wird eine Methode auf föderierter Ebene aufgerufen, kann sie je nach Zielsystem verschiedene Operationen auslösen. Wird auf eine solche „mehrdeutige" Methode eine ECA-Regel definiert, müssen bei einer Übersetzung anstatt nur eines lokalen Ereignisses mehrere verschiedene lokale Regeln zur Ereigniserkennung aufgestellt werden. Alternativ könnte der Aufruf dieser Methode auch durch „Method-Wrapping" (s. Abschnitt 3.2) erkannt werden. Zur Überprüfung der Integritätsbedingung müssen aber die entsprechenden lokal auftretenden Ereignisse erkannt

werden, um Konsistenzverletzung durch lokal laufende Anwendungen zu erkennen. Bei der Nutzung des „Method-Wrapping" für auf föderierter Ebene ausgeführte Anwendungen entsteht die folgende Situation: Das Ereignis zur Auslösung der ECA-Regel kann eines der lokal eingetretenen Ereignisse oder der Aufruf der Methode auf föderierter Ebene sein. Wenn man genau ein lokal eintretendes Ereignis spezifizieren will, muß man seine Zuordnung zu einer lokalen Datenbank (bzw. zu einem Datenbanksystem, das mehrere Datenbanken enthalten kann) vornehmen. Mit `DB1:e1` wird nun die Zuordnung eines bereits übersetzten, lokalen Ereignisses zu seiner Datenbank ausgedrückt. Das Ereignis `e1` kann konkret ein SQL-Befehl, z.B. ein komplettes `UPDATE`-Statement für eine relationales Komponentendatenbanksystem oder eine C++-Funktion, wie z.B. `class::method()` im Fall eines objektorientierten Komponentendatenbanksystems, sein.

Die Abbildung global definierter Regeln auf lokale und globale Regeln wird nun beschrieben. Es wird angestrebt, vorhandene aktive Mechanismen der Komponentendatenbanksysteme zu nutzen. Dies bedeutet, daß aus global definierten Regeln lokale Regeln abgeleitet werden müssen.

Zunächst wird ein Vorschlag für ein Verfahren der Regelumsetzung unterbreitet und dann an einem Beispiel durchgeführt. Zur Notation des Algorithmus werden noch einige Festlegungen getroffen:

- Mit `DBx:e1` eop_1 ... eop_{n-1} `DBx:en` ist hier ein komplexes Ereignis gemeint, das sich aus den Ereignissen `e1,..,en` zusammensetzt. Jedes dieser Ereignisse ist einer Datenbank, hier `DBx` genannt, wobei `DBx` eine beliebige Komponentendatenbank sein kann. Die einzelnen primitiven Ereignisse sind dabei wie im Abschnitt 3.2 beschrieben mittels der Ereignisoperatoren $eop_1,...,eop_{n-1}$ miteinander verknüpft. Teile von Bedingungen und Aktionen werden analog hierzu ihrer Datenbank zugeordnet.

- `LR_n_DBm_<regelname>` sei die n-te, von der durch den Regelnamen bezeichneten globalen Regel *abgeleitete* lokale Regel auf der lokalen Datenbank m.

- `GR_n_<regelname>` sei die n-te, von der durch den Regelnamen bezeichneten globalen Regel *abgeleitete* globale Regel.

- Die Aktion `signal(Regel_Mgr_<regelname>,DBx:ey)` dient der Benachrichtigung anderer Regelmanager. Hier wird das Ereignis `ey`, das auf der Datenbank `DBx` eingetreten ist, dem Regelmanager für die durch den Regelnamen bezeichnete ECA-Regel mitgeteilt.

Algorithmus zur Ableitung lokaler Regeln aus global definierten ECA-Regeln

1. Übersetze Ereignis, Bedingung und Aktion mittels Katalogzugriff auf föderierter Ebene.

Hierbei wird die Zuordnung von Teilereignissen, Teilbedingungen und Teilaktionen zu den einzelnen Datenbanken mit ermittelt. Außerdem weiß man nach dieser Übersetzung, ob ein relationales oder ein objektorientiertes System betroffen ist. Es ist damit auch bekannt, welche aktiven Mechanismen das betreffende System unterstützt. Wenn in föderierten Umgebungen einem Ereignis auf föderierter Ebene mehrere Ereignisse auf verschiedenen Komponentendatenbanksystemen zugeordnet sind, müssen diese Ereignisse alternativ verknüpft werden.

Die folgende allgemeingültige global definierte, jedoch schon mittels Metainformationen angereicherte Regel ist nun gegeben:

```
RULE      <regelname>
EVENT          DBx:e1 eop_1 DBx:e2 eop_2 ... eop_n-1 DBx:en
CONDITION      DBx:c1 cop_1 DBx:c2 cop_2 ... cop_n-1 DBx:cm
ACTION         DBx:a1;DBx:a2;...;DBx:ak;
```

2. Für jedes lokale primitive Ereignis `DBx:ey` generiere lokale Regel der Form:

```
RULE      LR_n_DBx_<regelname>
EVENT          ey
ACTION         signal(Regel_Mgr_<regelname>,DBx:ey)
```

auf der jeweils zugehörigen, aktiven Datenbank. Die eventuelle Umsetzung in einen Trigger für ein relationales Komponentensystem wird später behandelt.

3. Generiere neue globale Koordinationsregel der Form:

```
RULE      GR_<regelname>
EVENT          signal(Regel_Mgr_GR_<regelname>,DBx:e1) eop_1
               signal(Regel_Mgr_GR_<regelname>,DBx:e2)
               eop_2 ... eop_n-1
               signal(Regel-Mgr_GR_<regelname>,DBn:em)
CONDITION      DBx:c1 cop_1 DBx:c2 cop_2 ... cop_n-1 DBx:cm
ACTION         DBx:a1;DBx:a2;...;DBx:ak;
```

Diese Regel dient der Komposition der zusammengesetzten Ereignisse und enthält auch die Bedingung und die Aktion der Originalregel. Es also wird keine Abbildung von Bedingung und Aktion auf die lokalen Regeln durchgeführt. Warum dies nicht erfolgen darf, wird im Zusammenhang mit Optimierungsbemühungen für diesen Algorithmus im Abschnitt 4.5 besprochen. Die bei einer solchen Regelumsetzung und bei der Umsetzung von Konsistenzbedingungen entstehenden Metadaten müssen im Katalog des föderierten Systems verwaltet werden, damit bei einer Deaktivierung, Änderung oder Aufhebung einer

Konsistenzbedingung bekannt ist, welche lokalen ECA-Regeln oder Trigger nun ebenfalls deaktiviert, geändert oder entfernt werden müssen.

4.4.2 Trigger als lokale Regeln

Die Verwendung der von einigen relationalen Datenbanksystemen angebotenen Trigger ist nur dann möglich, wenn sie eine externe Funktion aufrufen können oder durch sonstige Mechanismen Meldungen über die Grenzen ihres Datenbanksystems ermöglichen. So ist es durchaus denkbar, über explizites Sperren von Datenbankobjekten eine Signalisierung vorzunehmen. Man kann in einer neu zu definierenden Tabelle Informationen zu lokalen Ereignissen speichern, und für den Zugriff sperren. Die Aufhebung einer solchen expliziten Sperre zeigt dem globalen Regelmanager (der diese Informationen jetzt lesen kann) an, daß das dort beschriebene Ereignis eingetreten ist. Der Aufruf einer externen Methode ist aus Gründen der Effizienz dem jedoch vorzuziehen. Im relationalen Datenbanksystem ORACLE kann ein Trigger innerhalb seines PL/SQL-Blockes keine externe Funktion aufrufen. In dem ORACLE-Anwendungsprogrammierungswerkzeug SQL-FORMS kann ein FORMS-Trigger ein externes Programm mit dem Befehl `USER_EXIT` aufrufen.

Die Konvertierung einer lokalen E(C)A-Regel kann recht einfach durch Einsetzen ihrer Bestandteile in eine Triggerdefinition (siehe Abschnitt 3.1.1) erfolgen. Es ist lediglich zu überlegen, welche variablen Teile des Triggers durch welche Angaben in der Regel ersetzt werden können, oder welche Bestandteile aufgrund der Heterogenität der Datenbankmanagementsysteme oder Datenmodelle durch ihre entsprechenden Komponenten für das Komponentensystem ersetzt werden müssen.

Die variablen Teile einer zur Propagierung lokal eintretender Ereignisse benutzten E(C)A-Regel sind der Ereignis- und der Aktionsteil. Es muß nun ermittelt werden, mit welchen Mitteln oder durch welchen SQL-Befehl eine Methode auf einer relationalen Datenbank arbeitet. Diese Informationen werden im Katalog des föderierten Datenbanksystems verwaltet. So kann zum Beispiel die Methode

```
setProdDate(serNr,typenBez)
```

durch den SQL-Befehl

```
UPDATE lagerung SET ProdDatum=sysdate
where SerienNr=serNr and Typ=typBez
```

ersetzt werden.

Die entsprechende Regel zur Meldung dieses Ereignisses könnte dann folgendermaßen gestaltet sein:

```
RULE   LR_1_DB1_Beispiel
```

```
EVENT       setProdDatum(sernr,typenbez)
ACTION      signal(global_Regel_Mgr,DB1:e1)
```

Diese Regel muß dann bei Verwendung des oben beschriebenen SQL-Statements in den nun folgenden ORACLE-Trigger umgesetzt werden.

```
CREATE TRIGGER LR1_1_Beispiel
 BEFORE oder AFTER
 UPDATE OF proddatum
 ON lagerung
 BEGIN
 <signalisiere dem globalen Regel_Mgr das Eintreten des Ereignisses>
 END
```

Es muß darüber hinaus eine Entscheidung getroffen werden, ob dieser Trigger als `before` oder `after` deklariert werden soll. Diese Entscheidung hängt davon ab, ob die Konsistenzsicherung im föderierten System nach einer optimistischen oder pessimistischen Strategie durchgeführt werden soll. Dazu ist ggf. die Deklaration einer globalen ECA-Regel als `before` oder `after` sowie die Angabe der Kopplungsmodi der Regel erforderlich (vgl. dazu Abschnitt 3.2).

4.5 Verbesserung des Ableitungsprozesses

Die Abarbeitung systemübergreifender ECA-Regeln verkompliziert sich im Vergleich zu der Regelabarbeitung in zentralen Systemen, da in föderierten Systemen ein Zusammenwirken mehrerer, auf verschiedenen Komponentendatenbanksystemen lokalisierten Regeln erfolgen muß. Man muß sich verdeutlichen, daß für *jedes* zu erkennende lokale primitive Ereignis eine lokal definierte Regel generiert werden muß. Aus diesem Grund wird im Abschnitt 4.5.1 eine Optimierung dieses Algorithmus bezüglich der Ereignisdetektion vorgenommen. Im Abschnitt 4.5.2 werden weitere Optimierungsmöglichkeiten untersucht. Im Anschluß daran wird eine Übersicht über den nun verbesserten Ableitungsalgorithmus gegeben.

4.5.1 Optimierung der Ereignisdetektion

Bei einer Ableitung globaler und lokaler Regeln aus global definierten ECA-Regeln wird insbesondere für jedes *primitive* Ereignis auf ein- und derselben Datenbank jeweils eine *eigene* Regel definiert. In gewissen, nachfolgend beschriebenen Fällen kann die Zahl der zu erzeugenden Regeln reduziert werden, indem man die Erkennung zusammengesetzter Ereignisse auf lokaler Ebene zuläßt. Wenn also im Ereignisteil einer global definierten ECA-Regel mehrere Ereignisse auf derselben Datenbank spezifiziert worden sind, sollte der in Abschnitt 4.4 behandelte Algorithmus einen Optimierungsschritt durchführen, der die Erkennung zusammengesetzter lokaler Ereignisse berücksichtigt. Um ein systemübergreifendes

komplexes Ereignis in lokale komplexe Ereignisse zu zerlegen, muß das gesamte globale komplexe Ereignis daraufhin untersucht werden, ob die Ereignisoperatoren eine Ableitung lokaler komplexer Ereignisse zulassen. Problematisch sind alle Operatoren, die eine zeitliche Reihenfolge implizieren, wie die Sequenz und die Operatoren, die Intervallgrenzen laut Tabelle 3.3 im Abschnitt 3.2 benötigen. So ist

`DB1:e1` **`and`** `DB2:e2` **`then`** `DB3:e3` nicht semantisch gleichwertig mit

`DB1:e2` **`then`** `DB1:e3` **`and`** `DB2:e2`.

Ein typisches Ereignis einer ECA-Regel zur Konsistenzsicherung ist das folgende:

Beispiel 4.5: Potentiell konsistenzverletzende Operationen sind die Erstellung und die Modifikation eines Objektes. Die Erstellung eines Objektes sei das Ereignis `e1`, die Modifikation sei das Ereignis `e2`. Beide Ereignisse sind alternativ verknüpft, d.h. beim Eintreten eines der beiden Ereignisse feuert die Regel. Das komplexe Ereignis der globalen ECA-Regel kann dann folgendermaßen geschrieben werden: `e1 or e2`. Die beiden Operationen können für semantisch gleichwertige Objekte auf zwei verschiedenen Datenbanksystemen, zu je eine der beiden Datenbanken `DB1` und `DB2` gehören, durch verschiedene Methoden ausgeführt werden. Es wird nun angenommen, daß dem Ereignis `e1` auf der Datenbank `DB1` das Ereignis `e1'`(möglicherweise ein SQL-Befehl), auf der Datenbank DB2 sei es `e1''`(z.B. eine C++-Funktion) entspricht. Analog dazu sind zu dem Ereignis `e2` auf der Datenbank `DB1` das Ereignis `e2'` und auf der Datenbank `DB2` das Ereignis `e2''` semantisch äquivalent. Das entstehende komplexe Ereignis ist dann das folgende:

`DB1:e1'` **`or`** `DB2:e1''` **`or`** `DB1:e2'` **`or`** `DB2:e2''`.

Hieraus können die beiden komplexen lokalen Ereignisse `ze1` und `ze2` mit

`ze1 = DB1:e1'` **`or`** `DB1:e2''`

und

`ze2 = DB2:e1'` **`or`** `DB2:e2''`

sinnvoll abgeleitet werden.

Bei der Verwendung erweiterter Syntaxbäume zur Ereignisdetektion (siehe dazu [Deu94]) bilden die primitiven Teilereignisse eines zusammengesetzten Ereignisses die Blätter des erweiterten Syntaxbaumes, während die Knoten die Operatoren enthalten. Die Wurzel des Baumes repräsentiert das gesamte zusammengesetzte Ereignis. Bildlich gesprochen, können Teilbäume, deren Blätter Ereignisse auf ein- und derselben Datenbank enthalten, die komplexen Teilereignisse für die abzuleitenden lokalen Regeln werden.

Unter Beachtung dieser Sachverhalte kann nun die Wirkung der Optimierung an einem Beispiel gezeigt werden. Der Optimierungsschritt hat zur Folge, daß die Anzahl neu entstehender lokaler Regeln auf das nötige Maß begrenzt wird, wie das nun folgende Beispiel 4.6 zeigt:

Beispiel 4.6: Die Optimierung der Umsetzung einer globalen Regel durch Zulassung der Erkennung komplexer Ereignisse auf lokaler Ebene wird an einer Beispielregel vollzogen. Die folgende global definierte Regel ist hierzu gegeben:

```
RULE      Beispiel
EVENT         (DB1:e1 or DB1:e2) and (DB2:e1 or DB2:e2)
CONDITION     ...
ACTION        a1
```

Gemäß dem im Abschnitt 4 .4 beschriebenen Algorithmus würden die folgenden lokalen Regeln `LR1` bis `LR4` aus der gegebenen Beispielregel generiert werden:

```
RULE      LR_1_DB1_Beispiel
EVENT         e1
ACTION        signal(Regel_Mgr_GR_Beispiel,DB1:e1)

RULE      LR_2_DB1_Beispiel
EVENT         e2
ACTION        signal(Regel_Mgr_GR_Beispiel,DB1:e2)

RULE      LR_3_DB2_Beispiel
EVENT         e1
ACTION        signal(Regel_Mgr_GR_Beispiel,DB2:e1)

RULE      LR_4_DB2_Beispiel
EVENT         e2
ACTION        signal(Regel_Mgr_GR_Beispiel,DB2:e2)
```

Es wird noch die globale Koordinationsregel GR erzeugt:

```
RULE      GR_Beispiel
EVENT         (signal(Regel_Mgr_GR_Beispiel,DB1:e1) or
               signal(Regel_Mgr_GR_Beispiel,DB1:e2)) and
              (signal(Regel_Mgr_GR_Beispiel,DB2:e1) or
               signal(Regel_Mgr_GR_Beispiel,DB2:e2))
CONDITION     ...
ACTION        a1
```

Durch eine Optimierung bei der Regelübersetzung können die beiden *zusammengesetzten* Teilereignisse `(DB1:e1 or DB1:e2)` und `(DB1:e1 or DB1:e2)` aus dem globalen zusammengesetzten Ereignis abgeleitet werden. Diese Ereignisse werden jetzt als `ze1` und

ze2 (zusammengesetztes Ereignis 1 bzw. 2) bei der Weitermeldung zum globalen Regelmanager bezeichnet. Die Zulassung der Erkennung lokaler komplexer Ereignisse hat zur Folge, daß jetzt nur zwei (statt bisher vier) lokale Regeln, und zwar auf jeder Datenbank genau eine Regel, benötigt werden:

```
RULE   LR_1'_DB1_Beispiel
EVENT        e1 or e2
ACTION       signal(Regel_Mgr_GR_Beispiel,ze1)


RULE   LR_2'_DB2_Beispiel
EVENT        e1 or e2
ACTION       signal(Regel_Mgr_GR_Beispiel,ze2)
```

Die neue globale ECA-Regel ist dann die folgende Regel:

```
RULE   GR_Beispiel
EVENT        signal(Regel_Mgr_GR_Beispiel,ze1)
             and
             signal(Regel_Mgr_GR_Beispiel,ze2)
CONDITION    ...
ACTION       a1
```

Dieses Beispiel basiert übrigens auf einer zur Konsistenzsicherung benötigten Regel, was verdeutlicht werden kann, wenn e1 z.B. eine Erzeugungsoperation und e2 z.B. eine Änderungsoperation für ein Objekt anzeigt. Diese beiden Operationen sind dann die potentiell konsistenzverletzenden Operationen für die durch die hier betrachtete ECA-Regel zu überwachende (fiktive) Konsistenzbedingung.

4.5.2 Weitergehende Optimierungsversuche

Bisher wurden lokale Regeln nur zur Propagierung lokal eingetretener Ereignisse genutzt. Es soll nun untersucht werden, ob die lokalen Regeln auch Teile der Bedingung überprüfen oder Teile der Aktion ausführen sollen. Wenn man sich die Verarbeitung einer ECA-Regel in einem zentralen Datenbanksystem betrachtet, fällt auf, daß die Ereignisdetektion, die Bedingungsauswertung und (wenn nötig) die Aktion immer zeitlich nacheinander, aber jeweils als Einheit betrachtet, durchgeführt werden. Werden dagegen bei einer Abarbeitung global definierter ECA-Regeln in föderierten Datenbanksystemen Teile der Bedingung oder der Aktion bereits von lokalen Regeln getestet bzw. ausgeführt, so kann es passieren, daß erst nach einer lokalen Bedingungsauswertung die globale Ereignisdetektion abgeschlossen wird. Zu diesem Zeitpunkt muß jedoch die Bedingung nicht mehr erfüllt sein. Ein Beispiel soll dies verdeutlichen:

Beispiel 4.7: Eine Beispielregel wird durch erweitere Optimierung übersetzt. Diese „Optimierung" besteht darin, daß die verteilte Bedingung `c1 and c2` in die lokale Regel `LR1` mit aufgenommen wird. Aus der folgenden globalen ECA-Regel:

```
RULE   Test_Regel
EVENT       DB1:e1 and DB1:e2 then DB2:e3
CONDITION   DB1:c1 and DB1:c2
ACTION      ...
```

würden die globale Kompositionsregel

```
RULE   GR_Test_Regel
EVENT       signal (Regel_Mgr_GR_Falsche_Regel,ze1)
            then
            signal (Regel_Mgr_GR_Falsche_Regel,DB2:e3)
CONDITION   ...
ACTION      ...
```

mit `ze1=(DB1:e1 und DB1:e2)`, sowie die beiden lokalen Regeln:

```
RULE   LR_1_DB1_Test_Regel
EVENT       e1 and e2
CONDITION   c1 and c2
ACTION      signal(Regel_Mgr_GR_Test_Regel,ze1)
```

und

```
RULE   LR_2_DB2_Test_Regel
EVENT       e3
ACTION      signal(Regel_Mgr_GR_Test_Regel,DB2:e3)
```

abgeleitet.

Es wurde bei dieser Umsetzung die systemübergreifende Bedingung der lokalen Regel zugeordnet. Die Betrachtung der Verarbeitung der so entstandenen Regeln soll zeigen, ob hier eine semantisch korrekte Regelumsetzung erfolgt ist: Auf der Datenbank `DB1` werden die Ereignisse `e1` und `e2` detektiert, d.h. `LR_1` wird von ihrem lokalen Regelmanager gefeuert. Die lokale Bedingung sei erfüllt, so daß eine Meldung zum globalen Regelmanager erfolgt. Eine Datenbankoperation wird gestartet, die eines der für die Bedingungsauswertung wichtigen Attribute verändert und damit die potentielle Konsistenzverletzung vermeiden soll. Die Bedingung der globalen ECA-Regel ist zu diesem Zeitpunkt *nicht* mehr erfüllt. Irgendwann danach wird das primitive Ereignis `e3` auf der zugehörigen Datenbank `DB2` erkannt, so daß nun LR2 gefeuert wird. Daraufhin hat die abgeleitete globale Regel `GR` alle Signale der lokalen Regelmanager erhalten und `GR` feuert. Die Aktion wird ausgeführt, obwohl die Bedingung der Regel *nicht* mehr erfüllt ist. Bei einer korrekten Regelumsetzung wird die Bedingungsauswertung immer erst *nach* der *vollständigen* Ereignisdetektion durchgeführt, d.h. Ereignis, Bedingung und Aktion sind in sich abgeschlossene Teile. Eine Parallele zur

Atomaritätseigenschaft der Transaktionen ist erkennbar, woraus sich ergibt, daß zumindest die gesamte Aktion innerhalb einer Transaktion ausgeführt werden sollte.

Die Betrachtung dieser Beispielregel zeigt, daß bei einer Bedingungsauswertung bereits durch eine lokale Regel die Semantik der ursprünglichen global definierten Regel nicht erhalten wird. Es gibt lediglich eine Ausnahme: Enthält eine globale ECA-Regel ein Ereignis, das nur aus Ereignissen eines lokalen Datenbanksystems zusammengesetzt ist und zur Bedingungungsauswertung sind nur in einer zu diesem Datenbanksystem gehörenden Datenbank gespeicherten Daten erforderlich, kann diese Bedingung mit in die lokale Regel aufgenommen werden. Eine Bedingungsauswertung durch globale Mechanismen liefert jedoch dasselbe Ergebnis bei der Regelverarbeitung. Liegt ein globales systemübergreifendes komplexes Ereignis vor, so kann die Bedingungsauswertung nicht von einer lokalen Regel übernommen werden. Der Ableitungsalgorithmus stellt sich also nun folgendermaßen dar, wenn die aus Abschnitt 4.4.1 bekannte allgemeine global definierte ECA-Regel benutzt wird:

4.5.3 Optimierter Ableitungsalgorithmus

Nachfolgend wird der nun im Vergleich zu dem Algorithmus im Abschnitt 4.4.1 verbesserte Ableitungsalgorithmus verbal beschrieben.

Optimierter Ableitungsalgorithmus zur Ableitung lokaler und globaler Regeln aus global definierten ECA-Regeln

1. Übersetze die Regelbestandteile inklusive ihrer Zuordnung zu einer Datenbank.

2. Analysiere Ereignisteil der global definierten Regel. Dieser Ableitungsschritt gliedert sich in zwei Schritte:

   2.1 Extrahiere Komplexe Teilereignisse, die einer Komponentendatenbank zugeordnet sind.

   2.2 Stelle eine Tabelle für die zusammengesetzten lokalen Ereignisse `ze1...zen` nach dem folgenden Muster auf (auch dies sind Metainformationen):

Lokales zusammengesetztes Ereignis	Notation (ohne DB-Identifikatoren)
`ze1`	`e1 and e2`

3. Generiere die lokalen Regeln mit den folgenden beiden Teilschritten:

   3.1 Für jedes lokale primitive Ereignis DBx:ey generiere lokale Regel der Form

```
RULE      LR_z_DBx_<regelname>
```

```
EVENT          ey
ACTION         signal(Regel_Mgr_<regelname>,DBx:ey)
```

auf der jeweils zugehörigen, aktiven Datenbank.

3.2 Generiere lokale E(C)A-Regeln für lokale zusammengesetzte Ereignisse `DBx:zey` der Form

```
RULE   LR_z_DBx_<regelname>
EVENT        e1 op ... op en
ACTION       signal(Regel_Mgr_<regelname>,DBx:zey)
```

wobei das Ereignis der im Schritt 2.2 erstellten Tabelle entnommen werden kann.

4. Generiere neue globale Regel der Form:

```
RULE   GR_<regelname>
EVENT        signal(Regel_Mgr_GR_<regelname>,DB1:e1)
CONDITION    <originale Bedingung>
ACTION       <originale Aktion>
```

5. Setze lokale Regeln für relationale Komponentendatenbanksysteme in Trigger um.

Abschließend sollen noch einige Bemerkungen zu diesem Algorithmus folgen:

Für den Schritt 1 des Algorithmus gilt zusätzlich zu dem in Abschnitt 4.4.1 Gesagten eine Einschränkung: Bei der Benutzung eines Triggers als lokale Regel ist zu beachten, daß ein Trigger immer nur auf eine Datenbankoperation definiert werden kann. Deshalb ist eine Optimierung, die die lokale Erkennung komplexer Ereignisse voraussetzt, nicht anwendbar, wenn Trigger als lokale Regeln fungieren sollen. Sobald feststeht, daß ein Teilereignis auf einer relationalen lokalen Datenbank erkannt werden soll, ist von einer Optimierung für dieses Teilereignis (Schritt 2) abzusehen.

In Bezug zu Schritt 3 dieses Algorithmus ist anzumerken, daß die hier entstehenden EA-Regeln für den Fall, daß sie auf einem relationalen Komponentendatenbanksystem mittels Trigger realisiert werden soll, nur eine Zwischenstufe darstellen. Im Schritt 5 werden dann erst die endgültigen Trigger aus dieser E(C)A-Regel (einer ECA-Regel ohne Bedingung) generiert.

5 Probleme der Konsistenzsicherung in föderierten DB

Im Kapitel 4 wurden Mechanismen vorgeschlagen, mit deren Hilfe Konsistenzsicherung in föderierten Datenbanksystemen durchgeführt werden kann. In Kapitel 5 hingegen soll nun untersucht werden, wie diese Mechanismen mit den Eigenschaften föderierter Datenbanksysteme, insbesondere des verschiedenen Formen der Autonomie harmonieren. Einschränkungen bezüglich dieser Autonomieforderungen der Komponentendatenbanksysteme werden zur Unterstützung einer Konsistenzsicherung nicht vermeidbar sein, da sie in dem in dieser Arbeit betrachteten Konsistenzsicherungssystem gegensätzliche Sachverhalte darstellen (genaueres dazu in Abschnitt 5.2). Im Abschnitt 5.1. werden Betrachtungen zu einem erweiterten bzw. abgeschwächten Konsistenzbegriff für föderierte Datenbanksysteme durchgeführt. Daran anschließend wird die Konsistenzsicherung in föderierten Datenbanksystemen unter dem Aspekt der Teilautonomie der Komponentensysteme betrachtet. Im Abschnitt 5.3 wird der Einfluß der Kommunikationsdauer der Komponentensysteme untereinander auf die Erkennung systemübergreifender komplexer Ereignisse untersucht. Ein ansatzweiser Lösungsvorschlag zur Elimination dieses Einflusses wird unterbreitet.

5.1 Konsistenzbegriff in föderierten Datenbanksystemen

Nach [Vos94] ist eine (zentralisierte) Datenbank konsistent, wenn die in ihr enthaltenen Daten alle Konsistenzbedingungen erfüllen. Demzufolge ist eine *föderierte* Datenbank konsistent, wenn alle in ihr existierenden Konsistenzbedingungen (sowohl lokale als auch globale Bedingungen) erfüllt sind.

Globale Konsistenzsicherung in föderierten Datenbanksystemen ist ein hoher Anspruch. Es muß gewährleistet werden, daß in einem System kooperierender, heterogener, autonomer Teilsysteme systemübergreifende Bedingungen eingehalten werden. Dies bedeutet, daß ein Konsistenzsicherungsmechanismus immer dann, wenn eine Konsistenzbedingung zu überprüfen ist, Zugriff auf Daten in Komponentensystemen beansprucht. Außerdem müssen zur flexiblen Definition von globalen Konsistenzbedingungen entweder die Hinzufügung deklarativer Konsistenzbedingungen direkt im Schema oder die Definition neuer Regeln bzw. Trigger auf aktiven Komponentendatenbanksystemen möglich sein. Werden ECA-Regeln zur Überwachung lokaler Konsistenzbedingungen oder zur Ereigniserkennung benutzt, ist bei der Verwendung des „Method Wrapping" zur Realisierung einer universellen Ereignisdetektion die Neuübersetzung aller auf den betreffenden Daten arbeitenden Anwendungsprogramme erforderlich (sofern nicht ein solches „Wrapping" bereits erfolgt ist).

Deshalb ist zu überlegen, ob globale Konsistenzsicherung jederzeit zweckmäßig oder sogar möglich ist. Kann man überhaupt sicherstellen, daß ein föderiertes System mit teilautonomen Komponentensystemen stets konsistent ist?

Wenn die globale Konsistenz nicht ständig gewährleistet werden kann, aber auf Konsistenzbedingungen im föderierten Datenbanksystem nicht verzichtet werden soll, gibt prinzipiell zwei Möglichkeiten für einen Kompromiß. Es ist zum einen möglich, die Autonomie der Komponentensysteme zu beschränken. Alternativ kann auch der Begriff der Konsistenz abgeschwächt werden.

Zur Einschränkung des Begriffes der Konsistenz eines föderiertes Systems gibt es zwei Möglichkeiten: Es kann sich der Charakter einer Konsistenzbedingung ändern, so daß im Verlauf der Arbeit mit den gespeicherten Daten unterschiedliche Reaktionen auf Konsistenzverletzungen erfolgen sollen (näheres hierzu in Abschnitt 5.4).

Solche eingeschränkte Konsistenz tritt nach [ABH+94] bei Entwurfs- und Konstruktionsprozessen auf, die zudem lang andauernde Prozesse sind und dementsprechend lange Transaktionen fordern. In den hier auftretenden Anwendungen werden globale Inkonsistenzen nicht nur geduldet, sondern im Anfangsstadium eines Entwurfs oft sogar als Normalfall betrachtet. In diesem Fall tritt das in objektorientierten Datenbanksystemen bekannte Problem der Versionierung der Daten auf, d.h. ein Objekt existiert in verschiedenen Versionen.

Die zweite Möglichkeit der Einschränkung des Konsistenzbegriffes besteht darin, die Einhaltung globaler Konsistenzbedingungen nur zeitweise zu garantieren. Diese temporäre Konsistenz (oder Inkonsistenz) wird auch von [CGW94] vorgeschlagen. Die Gültigkeit von Konsistenzbedingungen wird dort durch Zeitintervalle begrenzt oder durch ein Flag angezeigt.

Es kann weiterhin eingeschränkt werden, durch welchen Verursacher bedingte Konsistenzverletzungen global behandelt werden sollen. So können u.U. nur durch global aufgerufene Datenbankoperationen verursachte Konsistenzverletzungen zeitlich gebunden überwacht werden, wenn eine Überwachung lokal laufender Anwendungen nicht möglich ist. Eine Konsistenzverletzung durch ein solches von der Überwachung ausgeschlossenes Komponentensystem kann dann durch lokale Konsistenzsicherungsmechanismen, jedoch nicht zeitlich gebunden, erkannt werden.

5.2 Probleme bei der Konsistenzsicherung durch Autonomieforderungen

Die Konsistenzsicherung in föderierten Systemen und die Ausführungsautonomie lokaler Datenbanken bilden in dem in dieser Arbeit beschriebenen System zur Konsistenzsicherung in föderierten Systemen konträre Sachverhalte. Warum das so ist, wird bei Betrachtung der Architektur eines föderierten Datenbanksystems deutlich. In einem föderierten Datenbanksystem gibt es bekanntlich mehrere Komponentendatenbanken, auf denen zusätzlich zu globalen Anwendungen lokal ausgeführte Anwendungen laufen.

Lokale Anwendungen führen Operationen auf den lokalen Datenbanken aus. Diese Datenbankoperationen auf den lokalen (objektorientierten) Datenbanken werden mittels Funktionsaufrufen, z.B. in C++, durchgeführt. Wenn auf den Komponentendatenbanken nur mittels solcher Funktionsaufrufe gearbeitet wird, erfolgt die Ereignisdetektion zur Realisierung aktiven Verhaltens mittels „Method-Wrapping” (siehe Abschnitt 3.2). Im Gegensatz dazu können die in relationalen Datenbanksystemen durchzuführenden Operationen durch eine Komponente des Datenbankmanagementsystems selbst erkannt werden. Das „Method-Wrapping” erfordert eine Neuübersetzung der Quellcodes aller Anwendungen. Eine Neuübersetzung der Anwendungsquelltexte ist nur dann nötig, wenn diese Anwendung noch keine aktiven Fähigkeiten besitzt, also noch nicht einem „Method-Wrapping” unterzogen wurde.

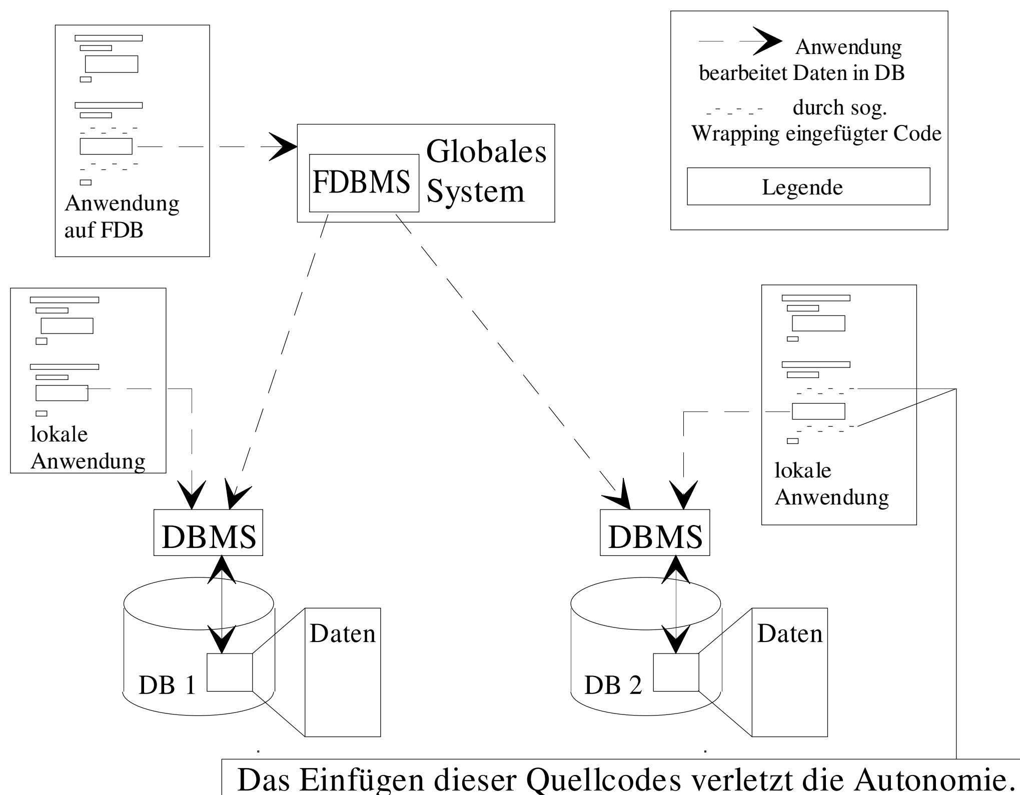

Abb. 5.1: Lokale und globale Datenbankanwendungen auf lokalen Datenbanken

Eine Neuübersetzung des Quelltextes einer lokalen Anwendung stellt jedoch einen Eingriff in die Autonomie (hier die Designautonomie - siehe Abschnitt 2.1) des zugehörigen Datenbanksystems dar. Gründe für solche Autonomieforderungen sind das Nichtvorhandensein der Anwendungsquelltexte, (die dann auch nicht neu übersetzt werden können) oder der Wunsch, einmal laufende Anwendungen nicht zu verändern.

Eine Einschränkung der Konsistenzsicherung (z.B. in Bezug auf die Art der Reaktion auf eine Konsistenzverletzung) zum Vorteil der Autonomie (z.B. der Ausführungsautonomie) kann gefährliche Dateninkonsistenzen nach sich ziehen, so daß dieser Weg bei (für eine Unternehmung) besonders lebenswichtigen Daten ungeeignet erscheint. Es ist dann möglich, daß lokale Anwendungen, die von der globalen Konsistenzsicherung nicht zeitlich überwacht werden, durch die Erzeugung inkonsistenter Daten die Arbeit des gesamten föderierten Datenbanksystems empfindlich stören.

Die Teilautonomie der Komponentensysteme hat auch dann eine hohe Priorität, wenn die lokale Zugänglichkeit zu den Daten sehr wichtig ist, und eine globale Bereitstellung der Daten oder eine globale Konsistenzsicherung von untergeordneter Bedeutung ist. Zu große Einflußnahme des globalen Systems hat dann Probleme, z.B. Performanceverluste auf lokaler Ebene zur Folge. Föderierte Systeme, in denen eine globale Konsistenzsicherung von untergeordneter Bedeutung ist, werden dann aber keine Konsistenzsicherung in dem Sinne, daß eine föderierte Datenbank konsistent zu erhalten ist, fordern. Der Autonomie der Komponentendatenbanksysteme sollte also Vorrang eingeräumt werden, wenn die globale Datenbereitstellung der lokalen Datenzugänglichkeit untergeordnet ist.

Die Konsistenzsicherung steht dann im Vordergrund, wenn globale Inkonsistenzen die korrekte Funktion des föderierten Datenbanksystems behindern. Dies kann bei Datenflüssen durch mehrere Komponentensysteme auftreten, wie sie bei dem Beispiel der unternehmensweiten Föderation im Abschnitt 2.3 vorkommen. Das föderierte Datenbanksystem hat hier die Aufgabe, alle unternehmensrelevanten Daten zu verwalten. Diese Aufgabe könnte auch von einer entsprechend großen zentralisierten Datenbank übernommen werden, wenn diese die volle Funktionalität aller Komponentendatenbanksysteme aufweist (siehe dazu auch [RS94]).

5.3 Ereigniskomposition in föderierten Datenbanksystemen unter dem Einfluß der Kommunikationsdauer

Konsistenzsicherung in föderierten Systemen bedeutet auch, daß beim Überprüfen der Einhaltung globaler Konsistenzbedingungen Kommunikation zwischen den Teilsystemen stattfindet. Im allgemeinen sind die Kommunikationsverbindungen zwischen den Systemen unterschiedlich. Diese Verbindungen haben unterschiedliche Übertragungsgeschwindigkeiten. Solche Geschwindigkeitsunterschiede entstehen z.B. durch unterschiedliche Datenübertragungsmengen, unterschiedliche Verbindungsbelastung oder verschiedene Antwortzeitverhalten der Komponentensysteme. Dadurch entstandene Zeitdifferenzen können Einfluß auf die korrekte Ereigniskomposition aktiver Konsistenzsicherungsmechanismen haben. Es wird durch den Regelmanager auf föderierter Ebene ein komposites Ereignis erkannt, daß nicht eingetreten ist, oder es wird ein eingetretenes Ereignis nicht erkannt. Es kann z.B. passieren, daß Meldungen über das Eintreten lokaler Ereignisse in einer anderen Reihenfolge beim globalen Regelmanager eintreffen, als die Ereignisse eintraten. Da jedoch mit dem Eintreten eines Ereignisses schon Operationen an Daten ausgeführt sein können, ist das vom globalen Regelmanager zusammengesetzte Ereignis nicht korrekt. Der folgende Sachverhalt soll dies verdeutlichen:

Beispiel 5.1: Ein System wird betrachtet, zu dem die beiden Teildatenbanken DB1 und DB2 gehören. Es gibt nun eine Konsistenzbedingung, die bei einer bestimmten Operation auf DB1 und einer zeitlich danach stattfindenden anderen Operation auf DB1 getestet werden muß. Bei der Betrachtung der Kommunikationszeiten fällt auf, daß die Meldung zur Indikation von lokalen Operationen zum globalen System im Falle von DB1 3 Sekunden, im Falle von DB2 jedoch nur 1 Sekunde in Anspruch nimmt.

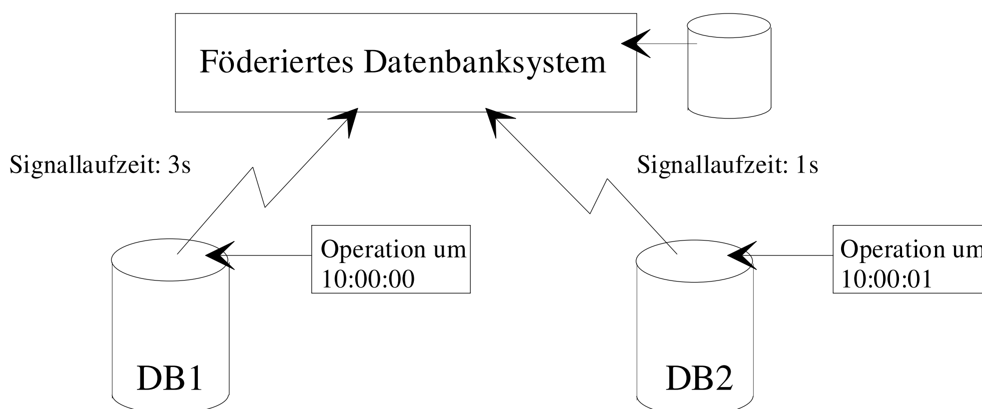

Abb. 5.2 Unterschiedliche Signallaufzeiten in einem verteilten System

Nun tritt folgender Betriebsfall auf:

1. Um 10:00:00 wird besagte Operation auf DB1 ausgeführt.

2. Um 10:00:01 erfolgt auf der Datenbank DB2 die Abarbeitung der dort relevanten Datenmanipulation.

3. Um 10:00:02 erhält das globale System Nachricht von dem zu DB2 gehörigen Datenbankmanagementsystem, erst 1 Sekunde danach vom Datenbankmanagementsystem der anderen Datenbank.

Die Operationen werden also nicht in derselben Reihenfolge vom föderierten System erkannt, in der sie tatsächlich stattgefunden haben. Es wird in diesem speziellen konstruierten Fall daraufhin *keine* Überprüfung der Konsistenzbedingung veranlaßt und im schlimmsten Fall gibt es nun eine Inkonsistenz im föderierten System.

Da das globale System die im allgemeinen zufällig ausfallende Kommunikationsdauer nicht kennt, kann es ohne geeignete Mechanismen diese nicht einkalkulieren. Das Beispiel zeigt aber auch, daß bei Signallaufzeiten im (Milli)Sekundenbereich eine derartige Situation recht selten auftritt. Gefährlich werden lediglich fast gleichzeitig auftretende Datenbankoperationen auf verschiedenen Komponentendatenbanken, wenn diese innerhalb eines komplexen Ereignisses in einem zeitlichen Zusammenhang stehen. Zur Lösung dieses Problems erscheint eine Vergabe von Zeitmarken (wie bei der Transaktionsverwaltung mittels Zeitmarken) für Ereignisse in föderierten Umgebungen praktikabel zu sein. In [JS94] wird der Vorschlag unterbreitet, Ereignisse als Transaktionen zu betrachten. Somit können auch traditionelle Sperrverfahren angewendet werden.

5.4 Zeitabhängige Gültigkeitsbetrachtung von Konsistenzbedingungen

Im Abschnitt 5.1 wurde der Begriff der Konsistenz für föderierte Datenbanksysteme behandelt. In diesem Abschnitt wird nun die zeitlich beschränkte Konsistenz betrachtet. In förderierten Systemen, bei denen der Leistungsfaktor eine nicht unwesentliche Rolle spielt, kann Konsistenzsicherung nicht ständig, sondern möglicherweise nur zu bestimmten Tageszeiten (z.B. nachts) durchgeführt werden (siehe [CGW94a]).

Wenn in einer Umgebung mit zeitlich langandauernden Prozessen, z.B. Konstruktions- und Entwurfsprozessen, Konsistenzbedingungen definiert werden, können diese Konsistenzbedingungen ihren Charakter ändern. Dies äußert sich darin, daß die Wichtigkeit der Geltung einer Bedingung dynamisch ist. Ein Entwurf ist zunächst weich, d.h. viele Designentscheidungen sind noch nicht endgültig. Daraus resultiert die Forderung einer flexiblen Reaktion auf Konsistenzverletzungen. In diesem Zusammenhang sprechen [ABH+94] von *harten* und *weichen* Konsistenzbedingungen. Bei sog. „weichen” Konsistenzbedingungen ist eine Korrektur der Inkonsistenz nicht zwingend nötig, während „harte” Konsistenzbedingungen nicht verletzt sein dürfen. Im Verlauf eines Entwurfsprozesses werden

meistens aus weichen Konsistenzbedingungen harte Konsistenzbedingungen, wie Beispiel 5.2 zeigen soll:

Beispiel 5.2: Mehrere Architekten arbeiten am Entwurf eines Gebäudes und zwei von ihnen verplanen zunächst ein- und denselben Raum für unterschiedliche Zwecke. Dies kann im Anfangsstadium des Entwurfs durchaus toleriert werden, wenn noch genügend Ausweichmöglichkeiten bestehen. Werden jedoch nach und nach alle Räume verplant, d.h. der Entwurf gewinnt an Festigkeit, muß auf diese Inkonsistenz hingewiesen werden, bis schließlich bei dem endgültigen Entwurf die Bedingung unbedingt eingehalten werden muß.

Entgegen dem bisher betrachteten Sachverhalt ist auch bei kontinuierlich durchgeführter Konsistenzprüfung und -sicherung die Möglichkeit zeitlicher Inkonsistenz gegeben. Es ist denkbar, daß diese Inkonsistenz von den Systemnutzern gewollt ist, um nicht durch ständige Warnungen in der Arbeit gestört zu werden. In solchen Situationen kann der Systemnutzer die Überprüfung von Konsistenzbedingungen unterbinden oder (wieder) zulassen.

Zur Realisierung einer flexiblen Reaktion auf Konsistenzverletzungen benötigt man die Möglichkeit, die Überprüfung global definierter Konsistenzbedingungen zeitweise zu unterbinden. Da globale Konsistenzsicherung durch Koordination von lokalen Konsistenzmechanismen realisiert werden soll, muß eine Deaktivierung der aus einer global definierten Konsistenzbedingung lokalen und globalen abgeleiteten Konsistenzbedingungen möglich sein. Es reicht nicht aus, eine globale Regel zu deaktivieren, es kann nötig sein, lokal abgebildete Teilbedingungen zu deaktivieren. Lokale, abgeleitete Konsistenzbedingungen können bei Deaktivierung der zugehörigen globalen Koordinationsregel überflüssig sein. Zeigt eine lokale Regel jedoch mehreren globalen Regelmanagern dasselbe Teilereignis an, so führt eine Deaktivierung dieser lokalen Regel zu Unwirksamkeit anderer Regeln, die dann bestimmte Teilereignisse nicht mehr gemeldet bekommen. Wenn wie in der hier vorgeschlagenen Architektur für eine Regelverarbeitung lokale Regeln Ereignisse jeweils nur einem globalen Regelmanager mitteilen, tritt dieses Problem nicht auf. Zur korrekten Deaktivierung oder Aktivierung global definierter Konsistenzbedingungen müssen aber auf jeden Fall bereits bei deren Umsetzung entsprechende Metainformationen in einem Katalog abgelegt werden (siehe Abschnitt 4.5).

Durch die Konsistenzprüfung selbst kann es auch Unterbrechungen in der Gültigkeit von Konsistenzbedingungen geben. Kann z.B. die Gültigkeit einer systemübergreifender Schlüsselbedingung aufgrund eingeschränkter Kommunikationswilligkeit eines Komponentensystems nicht verifiziert werden, ist keine eindeutige Aussage über die Richtigkeit dieser Bedingung möglich. Es ist noch zu klären, ob als Reaktion auf einen solchen (Ausnahme-)Fall die betreffende Bedingung als erfüllt oder verletzt propagiert werden soll. Dabei ist zwischen optimistischer oder pessimistischer Strategie zu wählen. Da ein Fall, wie der

hier beschriebene, meistens durch die fehlende Kooperativität eines einzelnen Systems auftreten kann, müßte die Schlüsselbedingung auf der Mehrzahl der beteiligten Komponentensysteme verifiziert werden. Es ist dann nur noch abzuwägen, wie hoch die Wahrscheinlichkeit der Verletzung der Bedingung durch dieses eine System sein kann (vielleicht enthält es nur einige wenige relevante Datensätze). Die Lösung dieser Art von Problemen kann auch mit dem in [GW95] vorgestellten Algorithmus zur Behandlung inkrementell wartbarer Konsistenzbedingungen erfolgen.

6 Schlußbetrachtung und Ausblick

In dieser Arbeit wurden zunächst einige wesentliche Aspekte föderierter Datenbanksysteme sowie die Grundlagen der Konsistenzsicherung in (zentralisierten) Datenbanksystemen aufgearbeitet. Insbesondere wurden hier diverse Konsistenzbedingungen für verschiedene Datenbanksysteme zusammengestellt und analysiert. Dazu wurden ORACLE 7 als Vertreter der relationalen Datenbanksysteme und ONTOS DB 3.0 als Vertreter der objektorientierten Datenbanksysteme auf ihre Unterstützung der Konsistenzsicherung hin untersucht. Außerdem erfolgte die Betrachtung des Konzeptes der zur Konsistenzsicherung nutzbaren ECA-Regeln aktiver Datenbanksysteme. Ferner wurden die klassischen Architekturen föderierter Datenbanksysteme behandelt und auf die Verträglichkeit mit dem Anspruch der globalen Konsistenzsicherung untersucht.

Eine Rahmenarchitektur eines möglichen globalen Regelverarbeitungssystems für föderierte Datenbanksysteme wird diskutiert. Das in dieser Arbeit betrachtete Regelverarbeitungssystem besteht aus lokalen und globalen Regelverarbeitungskomponenten, die hier Regelmanager genannt werden. Das Zusammenwirken dieser Regelmanager bei der Abarbeitung systemübergreifender ECA-Regeln wurde untersucht. Hieraus wurden Schlußfolgerungen für eine Übersetzung global definierter ECA-Regeln in durch die Regelmanager direkt zu verarbeitenden lokale und globale Regeln gezogen. Daran anschließend wurde die Verträglichkeit der zuvor diskutierten Mechanismen mit den wesentlichen Eigenschaften föderierter Datenbanksysteme untersucht.

Ein Hauptergebnis dieser Betrachtungen ist, daß effektive globale Konsistenzsicherung in föderierten Datenbanksystemen nur bei einer gleichzeitigen Schwächung der Autonomie der Komponentensysteme möglich ist. Darüber hinaus ist die angestrebte Verwendung bereits vorhandener lokaler Konsistenzsicherungsmaßnahmen nur eingeschränkt möglich. Außerdem wurde erkannt, daß eine Optimierung der Abarbeitung systemübergreifender ECA-Regeln nur bedingt möglich ist. Zur Beseitigung der Auswirkungen des Einflusses von Kommunikationsverzögerungen auf die Ereignisdetektion wurde ein Lösungsansatz unterbreitet. Überdies ist festgestellt worden, daß sich auch globale Konsistenzbedingungen mittels ECA-Regeln formulieren lassen.

Für die zukünftige Arbeit bleiben einige weiterhin bestehende Probleme zu lösen. Die Trigger des relationalen Datenbanksystems ORACLE 7 bieten z.Z. keine Möglichkeit externer Funktionsaufrufe zur Weitermeldung des Eintreten lokaler Ereignisse an eine globale Regelverarbeitungskomponente an. Dagegen ist diese Möglichkeit bei den Triggern des ORACLE-Tools SQL-FORMS vorhanden, weshalb die Möglichkeit einer Einbeziehung dieser FORMS-Trigger in das in dieser Arbeit diskutierte Konsistenzsicherungssystem für föderierte

Systeme zu untersuchen ist. Im Abschnitt 3.2 dieser Arbeit wurde angesprochen, daß ein enger Zusammenhang zwischen den beiden Kopplungsmodi einer ECA-Regel und den von einem Datenbanksystem unterstützten Transaktionsmodellen besteht. In diesem Zusammenhang muß geklärt werden, welche Kopplungsmodi für ECA-Regeln in föderierten Systemen sinnvoll sind und mit Hilfe welcher Transaktionsmodelle sie in föderierten Umgebungen realisiert werden können. Im Zusammenhang mit der Definition von ECA-Regeln in föderierten Systemen ist eine Untersuchung der Operatoren zur Verknüpfung von Teilbedingungen zu einer systemübergreifenden Bedingung einer globalen ECA-Regel nötig. Hier ist herauszufinden, ob die in dieser Arbeit genannten Operatoren ausreichen oder eine Erweiterung der vorhandenen Operatorenmenge, die z.B. logische und Vergleichsoperatoren umfaßt, um weitere Operatoren nötig ist. Bei der exemplarischen Aufstellung der Schemata des Beispielszenario im Kapitel 2 wurde deutlich, daß die Formulierung globaler Konsistenzbedingungen durch deklarative Beschreibungsmittel wünschenswert ist, weshalb weitergehende Betrachtungen von Mitteln zur Formulierung globaler Konsistenzbedingungen erfolgen sollten.

Thesen

1. Effektive globale Konsistenzsicherung in föderierten Datenbanksystemen ist nur bei einer gleichzeitigen Schwächung der Autonomie der Komponentensysteme möglich.

2. Die Verwendung bereits vorhandener lokaler Konsistenzsicherungsmaßnahmen zur globalen Konsistenzsicherung ist nur eingeschränkt möglich.

3. Globale Konsistenzbedingungen lassen sich mittels ECA-Regeln formulieren bzw. realisieren.

4. Eine Optimierung der Abarbeitung systemübergreifender ECA-Regeln ist nur bedingt möglich. Konkret bedeutet dies, daß von lokalen Regelverarbeitungskomponenten weder Bedingungsauswertungen noch andere Aktionen als die Propagierung eingetretener Ereignisse durchführen können.

5. Zur Beseitigung der Auswirkungen des Einflusses von Kommunikationsverzögerungen auf die Ereignisdetektion bei der Abarbeitung von ECA-Regeln in föderierten Systemen kann das aus der Transaktionsverwaltung bekannte Zeitmarkenverfahren benutzt werden.

Quellenverzeichnis

[ABH+95] K. Abramowitz, B. Boss, V. Hovestadt, J. Mülle, R. Sturm und P. C. Lockemann: *„Konsistenzüberwachung in Datenbanksystemen - Eine Anforderungsanalyse Anhand der Entwurfsbereiche Architektur und Schiffbau"*, In G. Lausen (Herausgeber): „Proc. Datenbanksysteme in Büro, Technik und Wissenschaft (BTW'95)", Dresden, S. 302-321. Springer-Verlag, Informatik aktuell, März 1995.

[BEK+94] R. Böttger, Y. Engel, G. Kachel, S. Kolmschlag, D. Nolte und E. Radeke: *„Enhancing the Data Openness of Framworks by Database Federation Services"*, Technical Report, Cadlab Paderborn, 1994.

[BFHK94] R. Busse, P. Fankhauser, G Huck und W. Klas: *„IRO-DB: An Object-Oriented Approach Towards Federated and Interoperable DBMS"*, In: „Proc. of the Int. Workshop on Advances in Databases and Information Systems (ADBIS'94)", Moscow, Russia, S. 178--186. Russian Academy of Sciences, Mai 1994.

[BFN94] R. Busse, P. Fankhauser und E. J. Neuhold: *„Federated Schemata in ODMG"*, In J. Eder und L. A. Kalinichenko (Herausgeber): „Extending Information Systems Technology - Proc. of the 2nd Int. East/West Database Workshop, Klagenfurt, Austria", S. 356-379. Springer-Verlag, Workshops in Computing, September 1994.

[BGS95] Y. Breitbart, H. Garcia-Molina und A. Silberschatz: *„Transaction Management in Multidatabase Systems"*, In W. Kim (Herausgeber): „Modern Database Systems", S. 573-591. ACM Press, 1995.

[Buc94] A. Buchmann: *„Active Object Systems"*, In A. Dogac, M. T. Öszu, A. Biliris und T. Selis (Herausgeber): „Advances on Object-Oriented Database Systems", S. 201-224. Springer-Verlag, 1994.

[BZBW95] A. P. Buchmann, J. Zimmermann, J. A. Blakeley und D. L. Wells: *„Building an Integrated Active OODBMS: Requirements, Architecture, and Design Decisions"*, In P. S. Yu und A. L. P. Chen (Herausgeber): Proc. of the 11th Int. Conf. on Data Engineering (ICDE'95), Taipei, Taiwan, S. 117-128. IEEE Computer Society, März 1995.

[CGW94] S. Chawathe, H. Garcia-Molina und J. Widom: *„Constraint Management in Loosely Coupled Distributed Databases”*, Technical Report, Department of Computer Science, Stanford University, 1994.

[CM93] S. Chakravarthy und D. Mishra: *„Snoop: An Expressive Event Specification Language for Active Databases”*, Technischer Bericht UF-CIS-TR-93-007, University of Florida, März 1993.

[CW92] S. Ceri und J. Widom: *„Production Rules in Parallel and Distributed Database Environments”*. In L.-Y. Yuan (Herausgeber): „Proc. of the 18th Conf. on Very Large Data Bases (VLDB'92), Vancouver, Canada”, S. 339-351. Morgan Kaufmann Publishers, August 1992.

[Day88] U. Dayal: *„Active Database Management Systems”*, in: Procedings of the 3rd International Conference on Data and Knowledge Bases, Jerusalem Israel, S. 150-169, Juni 1988.

[DBB+88] U. Dayal, B. Blaustein, A. Buchmann, S. Chakravarthy, D. Goldhirsch, M. Hsu, R. Ladin, D. McCarthy und A. Rosenthal: *„The HIPAC-Project: Combining Active Databases and Timing Constraints”*. ACM SIGMOD RECORD, 17(1):51-70, März 1988.

[Deu94] A. Deutsch: *„Detektion von Methoden- und kompositen Events im aktiven DBMS REACH”*, Diplomarbeit, Technische Hochschule Darmstadt, 1994.

[DHW95] U. Dayal, E. Hanson und J. Widom: *„Active Database Systems”*, In W. Kim (Herausgeber): „Modern Database Systems”, S. 434-456. ACM Press, 1995.

[GD93] S. Gatziu und K. R. Dittrich: *„Eine Ereignissprache für das aktive, objektorientierte Datenbanksystem SAMOS”*. In W. Stucky und A. Oberweis (Herausgeber): Proc. GI-Fachtagung „Datenbanksysteme in Büro, Technik und Wissenschaft (BTW'93), Braunschweig”, S. 54-73, Springer-Verlag, Informatik aktuell, Mai 1993.

[GGD94] S. Gatziu, A. Geppert und K. R. Dittrich: *„SAMOS: an Active Object-Oriented Database System”*, Technical Report 94.16, University of Zurich, 1994.

[GGF+95] G. Gardarin., S. Gannouni, B. Finance, P. Fankhauser, W. Klas, D. Pastre, R. Legoff und A. Ramfos: *„IRO-DB - A Distributed System Federating Object and Relational Databases”*, In O. Bukhres und A. K. Elmagarmid (Herausgeber): „Object-Oriented Multidatabase Systems - A Solution for Advanced Applications”, Prentice Hall, 1995.

[GPQ+94] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, J. Sagiv,J. Ullman und J. Widom: *„The TSIMMIS Approach to Mediation: Data Models and Languages (Extended Abstract)"*, Technical Report, Stanford University, 1994.

[GW95] *P. Grefen, J. Widom: „Integrity Constraint Checking in Federated Databases"*, Technical Report, Stanford University, 1995.

[JQ92] H. V. Jagadish und X. Qian: *„Integrity Maintinance in an Object-Oriented Database",* In L.-Y. Yuan (Herausgeber): „Proc. of the 18th Conf. on Very Large Data Bases (VLDB'92)", Vancouver, Canada, S. 469-480. Morgan Kaufmann Publishers, August 1992.

[JS94] H. V. Jagadish und O. Shmueli: *„Composite Events in a Distributed Object-Oriented Databases"*, In T. Öszu, U. Dayal, und P. Valduriez. (Herausgeber): „Distributed Object Management", S. 248-268, Morgan Kaufmann Publishers, 1994.

[Kot93] A. M. Kotz-Dittrich. *„Adding Active Functionality to an Object-Oriented Database System - a Layered Approach"*, In W. Stucky und A. Oberweis (Herausgeber): Proc. GI-Fachtagung „Datenbanksysteme in Büro, Technik und Wissenschaft" (BTW'93), Braunschweig, S. 54-73. Springer-Verlag, Informatik aktuell, Mai 1993.

[MY95] W. Meng und C. Yu: *„Query Processing in Multidatabase Systems"*, In W. Kim (Herausgeber): „Modern Database Systems", S. 551-572, ACM Press, 1995.

[ONT94] ONTOS Inc.: *„ONTOS DB 3.0 - Tools and Utilities Guide"*, ONTOS, Inc., 1994.

[ORA92] ORACLE Corporation: *„ORACLE 7 Server - SQL Language Quick Reference"*, December 1992.

[ÖV91] M. T. Öszu, P. Valduriez: *„Principles of Distributed Database Systems"*, Prentice Hall, 1991.

[Rad94] E. Radeke: *„Efendi: Federated Database System of Cadlab"*, Technical Report, University of Paderborn & Siemens Nixdorf, 1994.

[RS94] E. Radeke und M. H. Scholl: *„Federation and Stepwise Reduction of Database Systems"*, In W. Litwin und T. Risch (Herausgeber): Proc. of the 1st Int. Conf.

on Applications of Databases, Sweden, S. 381-399. Springer-Verlag, LNCS 819, Juni 1994.

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

[Stü93] G. Stürmer: *„ORACLE7 - Die verteilte semantische Datenbank"*. DBMS Publishing, 2. Auflage, 1993.

[TC95] C. Türker und S. Conrad: *„Aktive Mechanismen zur Konsistenzsicherung in föderierten Datenbanksystemen"*, In C. Eckert, H.-J. Klein und T. Polle (Herausgeber): 7. GI-Workshop „Grundlagen von Datenbanken", S. 148-152. Hildesheimer Informatik Berichte, Juni 1995.

[Tür94] C. Türker: *„Grundlagen des Regelmanagements eines aktiven Objektsystems"*, Diplomarbeit, Technische Hochschule Darmstadt, 1994.

[Vos94] G. Vossen: *„Datenmodelle, Datenbanksprachen und Datenbank-Management-Systeme"*, 2.Auflage, Addison Wesley, 1994.

[ZDDM94] J. Zimmermann, A. Deutsch, W. Dürhold und J. Marschner: *„Architektur des aktiven Datenbanksystems REACH"*, Technical Report, TH Darmstadt, 1994.

[Zim94] J. Zimmermann: *„Das aktive Objektsystem von REACH"*. In: Workshop der GI-Fachgruppe 2.5.1: Aktive Datenbanken, Hamburg, Springer-Verlag, Informatik aktuell, September 1994.

Abbildungsverzeichnis

Abb. 2.1: FDBS mit mehreren föderierten Schemata - lose gekoppelter Ansatz 14

Abb. 2.2: FDBS mit einem föderierten Schema - eng gekoppelter Ansatz 15

Abb. 2.3: Unternehmensübergreifende Föderation 17

Abb. 2.4: Unternehmensweite Föderation mit eng gekoppelter Architektur 19

Abb. 4.1: Lokale und globale Konsistenzbedingungen in einem FDBS.................................39

Abb. 4.2: Architekturvorschlag zur Verarbeitung von ECA-Regeln in FDBSen....................52

Abb. 5.1: Lokale und globale Datenbankanwendungen auf lokalen Datenbanken..................68

Abb. 5.2 Unterschiedliche Signallaufzeiten in einem verteilten System.................................81

Tabellenverzeichnis

Tabelle 3.1: Deklarative Konsistenzbedingungen in ORACLE 7 28

Tabelle 3.2: Konsistenzbedingungen des objektorientierten Datenbanksystem ONTOS.........31

Tabelle 3.3: Standartoperatoren zur Definition zusammengesetzter Ereignisse 33