Teil II:
Heterogene Datenbanksysteme

Integration heterogener Informationsquellen

Literatur: Stefan Conrad, Föderierte Datenbanksysteme, Springer Verlag, 1998.

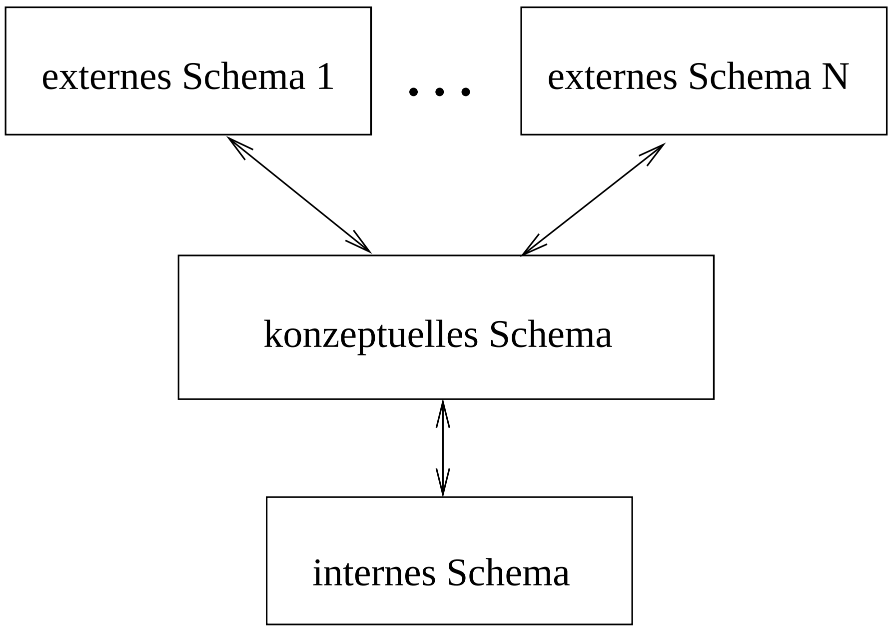
Ziel: (logische) Integration existierender Datenbanken.

- Integration von Systemen innerhalb eines Unternehmens (z.B. Workflow-Systeme; Data Warehouses);
  u.a., *Legacy*-Datenbanken, häufig noch Netzwerk- oder hierarchisches Datenmodell);
  Migration.
- *Global Information Sharing:* globale Verbindung von Datenbanken und Informationssystemen verschiedener Unternehmen/Organisationen;
- Grenzbereich: Nutzung von irgendwo (WWW) bereitgestellter Information.

Nebeneffekte

- Tools und Anwendungen müssen ggf. ebenfalls integriert werden
- globale Integritätsbedingungen
- globale Transaktionsverwaltung

Klassische 3-Ebenen-Architektur nach ANSI/X3/SPARC



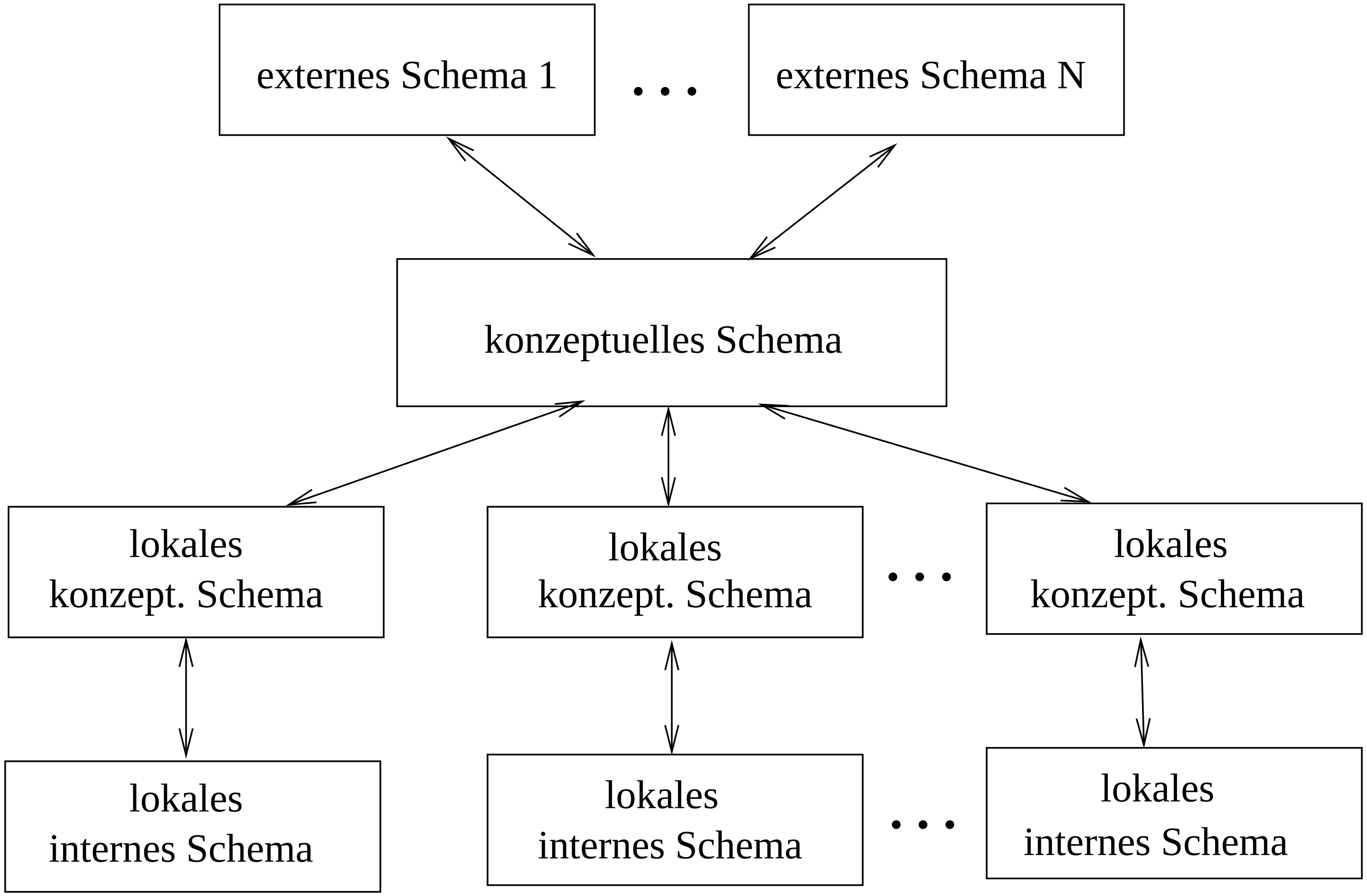
- Externe Schemata: Benutzersichten
- Konzeptionelles Schema: Abstrakte Beschreibung der Daten; umfaßt alle Benutzersichten
- Internes Schema: Implementierung des konzeptionellen Schemas für ein konkretes Datenbanksystem; enthält auch zusätzliche Datenstrukturen (Indexe etc)

$\Rightarrow$ Datenunabhängigkeit

Verteilte Datenbanksysteme

- Speicherung der Daten in verschiedenen Datenbanken, auf verschiedenen Rechnern.
- Verwaltung durch ein einziges Datenbanksystem
  Anwender hat den Eindruck, auf einem zentralen Datenbanksystem zu arbeiten.

4-Ebenen-Schema-Architektur für verteilte Datenbanksysteme:



Möglich: Redundanz zwischen den lokalen konzeptionellen Schemata

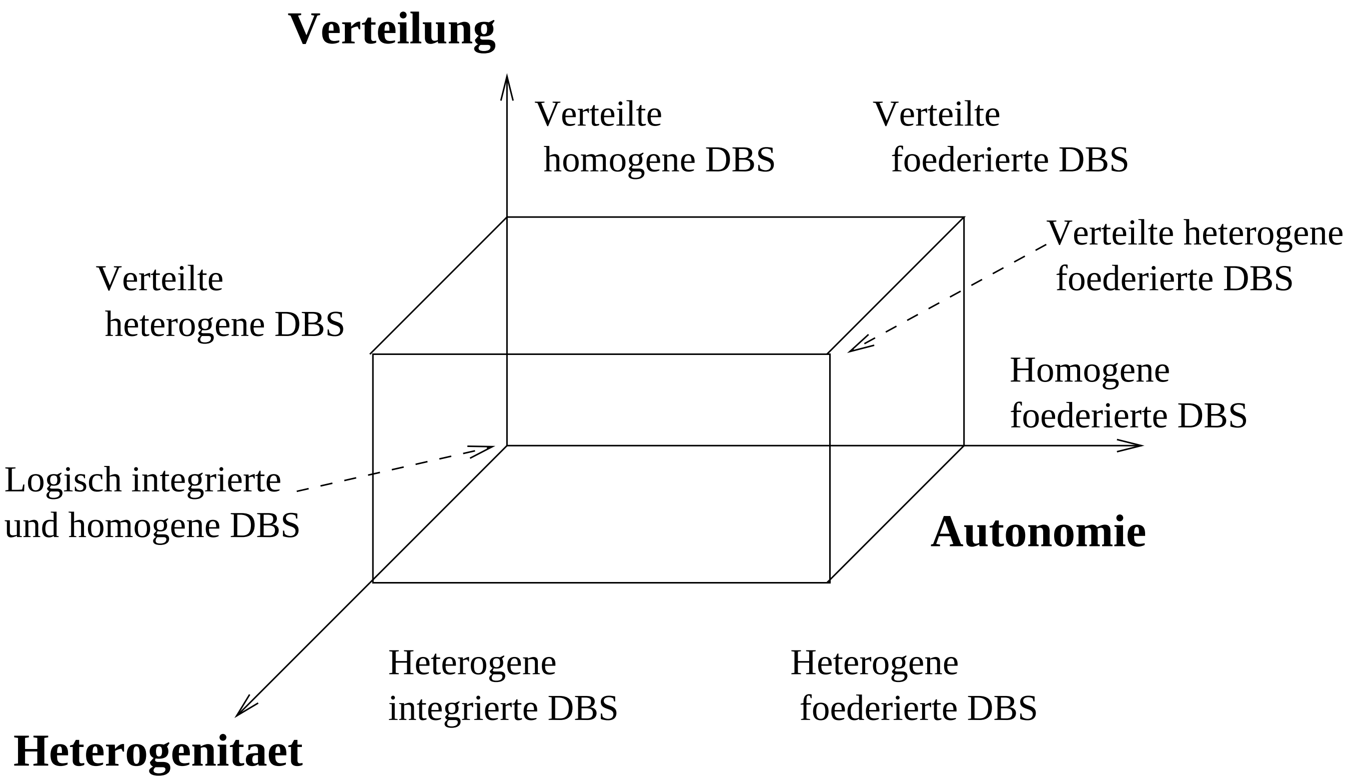
Multidatenbanksysteme

Verbund mehrerer Datenbankmanagementsysteme

- möglich: heterogene lokale Datenbanken:
  - unterschiedliche Datenmodelle (Netzwerk, RDBMS, OODBMS)
  - selbes Datenmodell, aber unterschiedliche Schemata
  - selbes Datenmodell, aber unterschiedliche DML
  - selber Sachverhalt unterschiedlich modelliert
- Autonomie der einzelnen beteiligten Datenbanken
  kann auch verschieden stark ausgeprägt sein, z.B.
  - Teildatenbanken einer Firma (Produktion, Verkauf),
  - Datenbanken von einer Firma und Zulieferern
  - Reisebüro: verwendet DB verschiedener Fluggesellschaften, Mietwagenfirmen etc. . . .
  - Nutzung von Datenbanken aus dem Web (ohne deren Wissen)

  ⇒ erfordert vollkommen unterschiedliche Konzepte, insbesondere der letzte Punkt sprengt die klassischen Vorstellungen.

Verteilte, Autonome, Heterogene Datenbanken



Föderierte Datenbanken

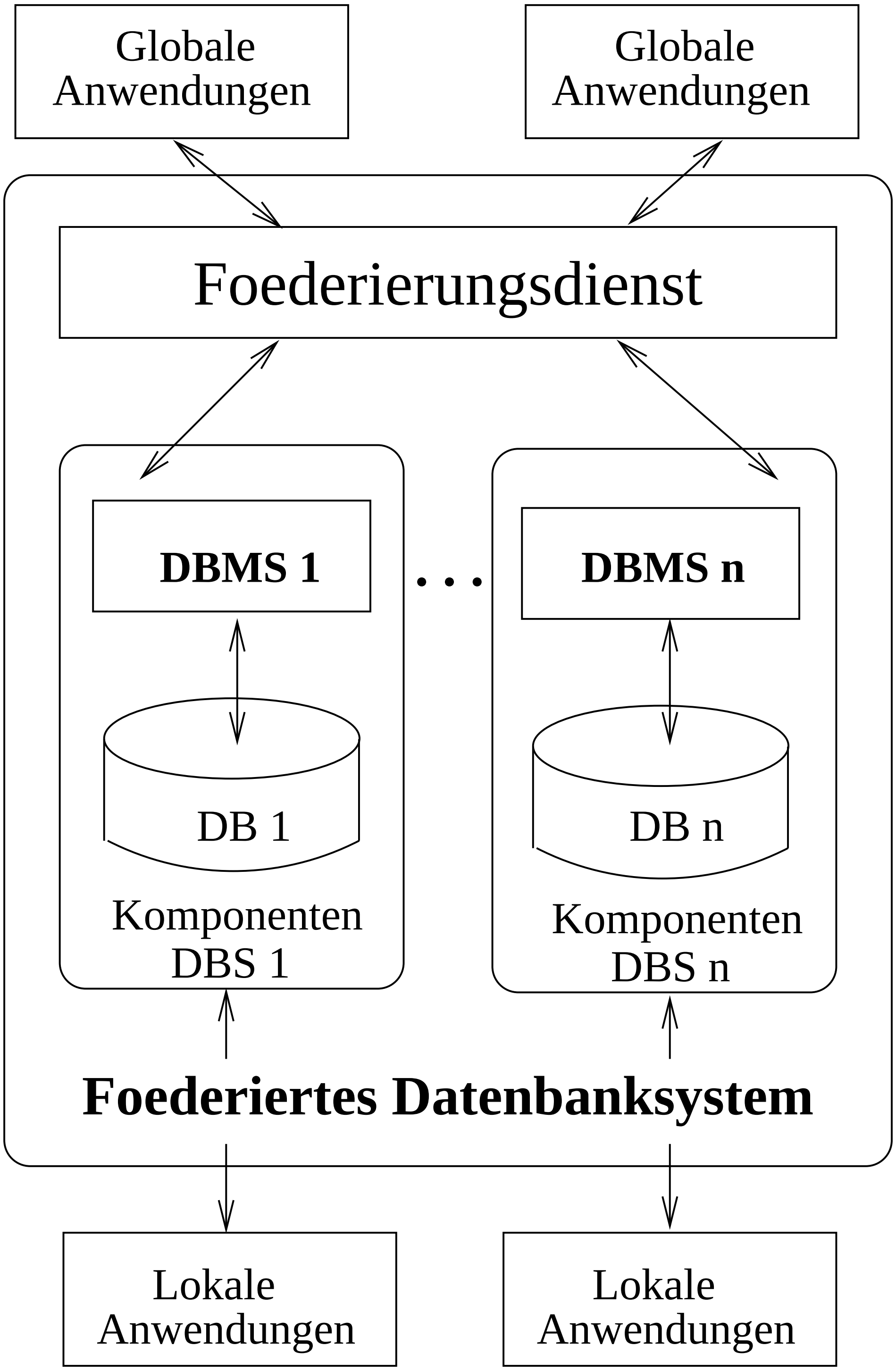
- *Föderation* als politischer Begriff bezeichnet ein Bündnis, d.h. einen vertraglichen Zusammenschluß von Staaten oder Teilstaaten.
- Ein *föderiertes Datenbanksystem* (FDBS) ist eine Menge von *kooperierenden autonomen* Datenbanksystemen; die Komponenten-Datenbanksysteme werden durch die Föderation nicht berührt.
- *nicht föderiertes Multi-DBS:* Komponentendatenbanken sind nicht autonom, vergleichbar zu einem verteilten DBS.

Kopplung

Enge Kopplung: Föderation wird durch einen entsprechenden Administrator zusammengestellt, überwacht, und die Funktionalität bereitgestellt.

Lose Kopplung: Benutzer stellt die Föderation selber zusammen und verwaltet sie
$\Rightarrow$ Benutzer muß den Föderierungsdienst/Mediator selber bereitstellen.

Allgemeine Architektur



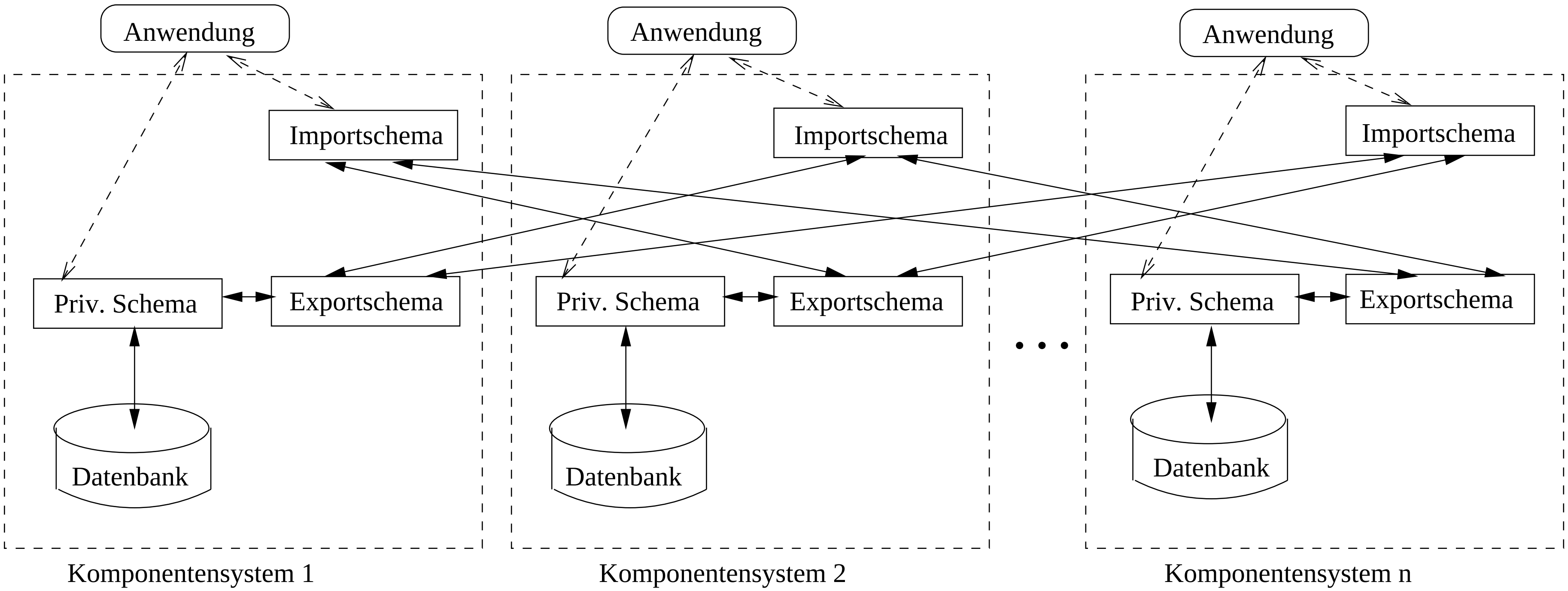
Allgemeine Architektur

- Lokale Anwendungen laufen unter *lokalen Schemata*
- *Föderierungsdienst* (*Mediator*) liefert eine globale Sicht auf ein globales Schema.

$\Rightarrow$ Abbildung der lokalen Schemata sowie der Daten auf das globale Schema notwendig.

Import-/Export-Schema-Architektur für föderierte Datenbanken

(Heimbigner, McLeod 1981 – eine der grundlegenden Arbeiten auf diesem Gebiet)



Import-/Export-Schema-Architektur

(Heimbigner, McLeod 1981)

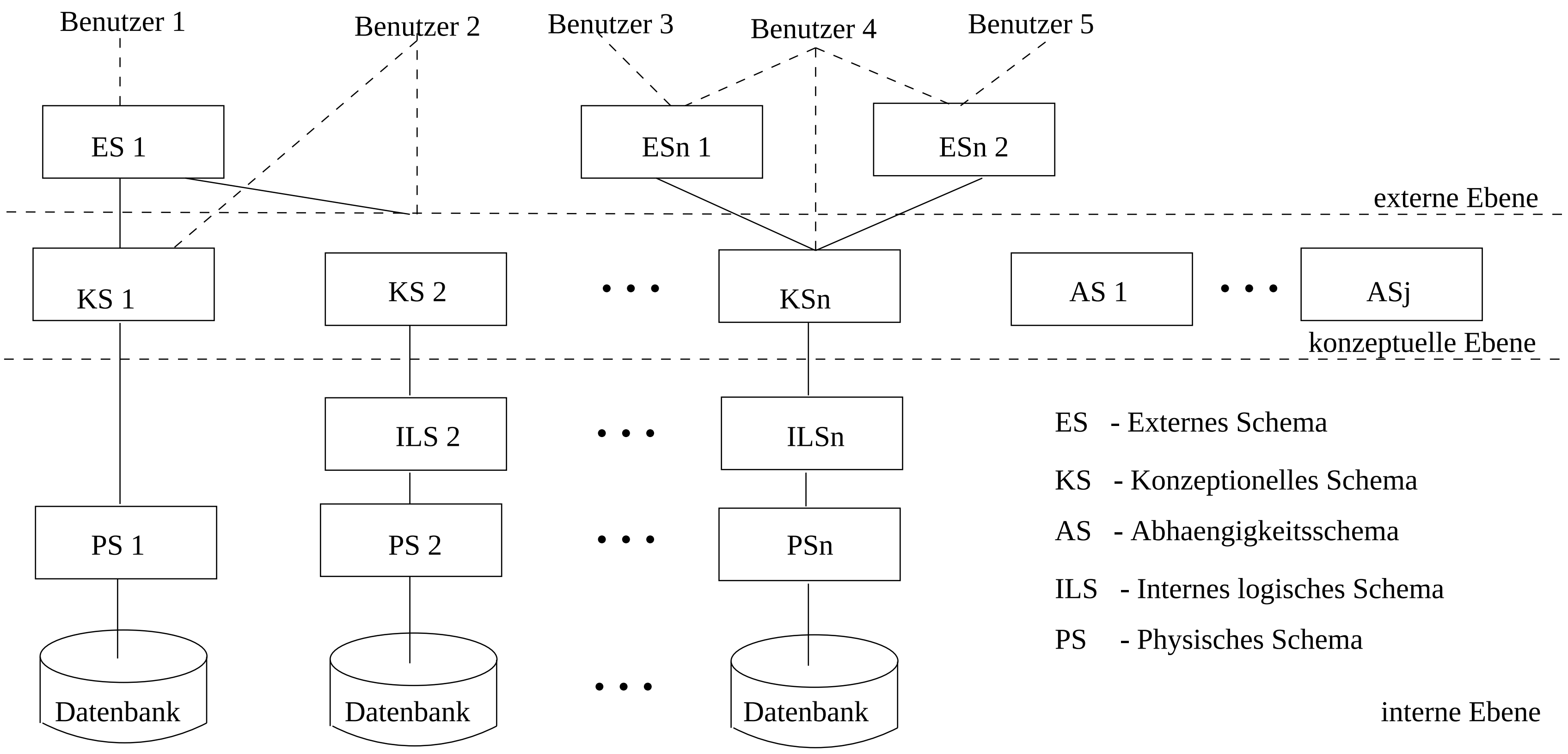
Privates Schema: Definiert die von der Komponente verwalteten Daten. Entspricht dem konzeptionellen Schema; zusätzlich Informationen über die Teilnahme an der Föderation.

Exportschema: Ausschnitt des privaten Schemas;
Definiert die der Föderation zur Verfügung gestellten Daten sowie Zugriffsrestriktionen für die beteiligten Komponentensysteme (nicht für bestimmte Nutzer).

Importschema: Definiert die von anderen Komponenten zu verwendenden Daten.

Abbildungen: zwischen Import- und Exportschema bzw. privatem und Exportschema bestehen Abbildungen, durch die Anfragen im Hinblick auf das betreffende Schema umformuliert werden.

Multidatenbank-Architektur (Litwin 1988)



Multidatenbank-Architektur

Physisches Schema: wie nach ANSI/SPARC.

Internes logisches Schema: wie konzeptionelles Schema nach ANSI/SPARC.

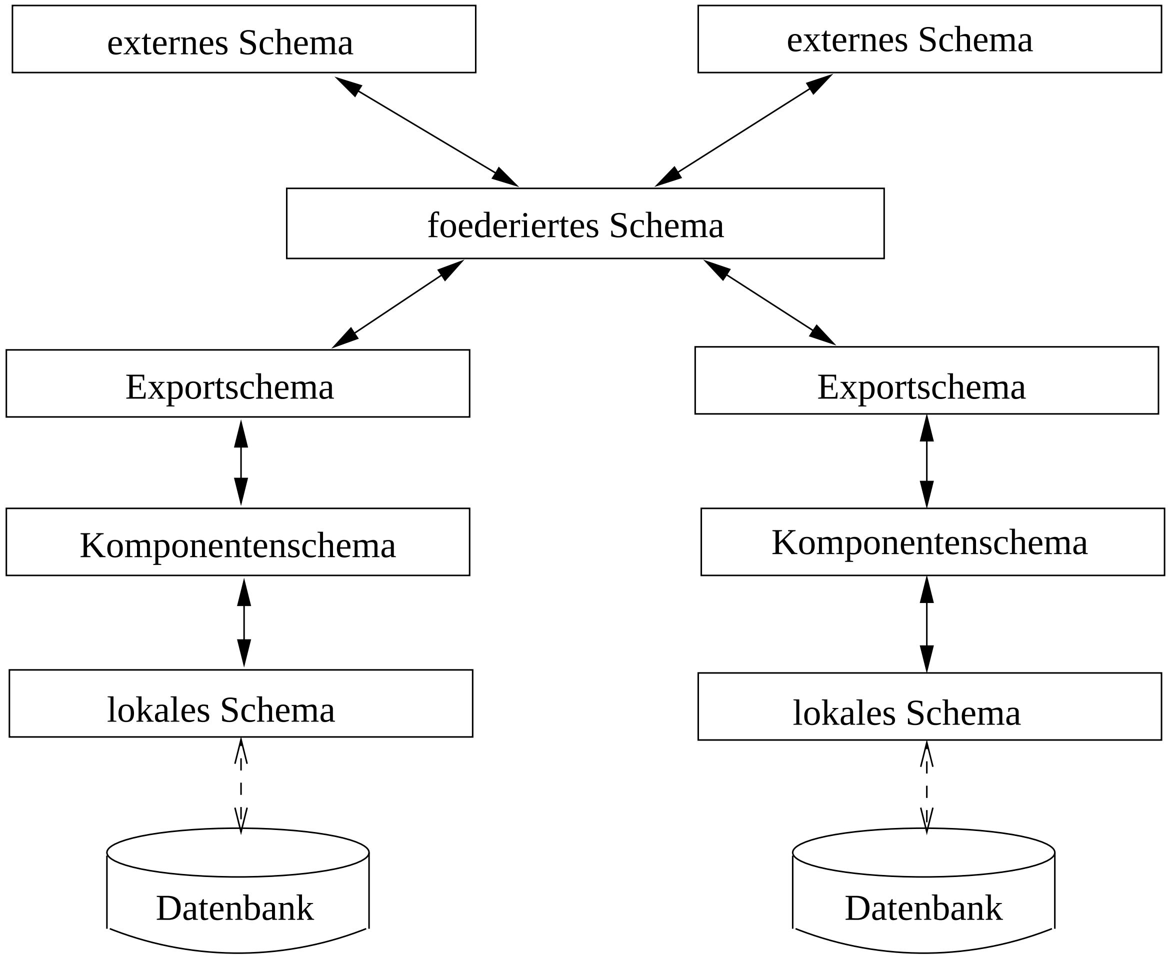
Konzeptionelles Schema: Definiert den für die Föderation zulässigen Anteil des internen logischen Schemas (entspricht externem Schema nach ANSI/SPARC). Alle konzeptionellen Schemata müssen ein gemeinsames Datenmodell und eine gemeinsame Datenbanksprache besitzen.

Externes Schema: Jeder Benutzer kann sich unter Verwendung einer *geeigneten Sprache* sein eigenes Schema aus den verfügbaren konzeptionellen Schemata der Komponenten definieren. Im Prinzip besteht auch die Möglichkeit, direkt auf diesen zu arbeiten.

Abhängigkeitsschema: Abhängigkeiten der Komponenten untereinander (*Interdatenbankabhängigkeiten*) werden vom Multidatenbanksystem überwacht.

Anforderungen an eine *geeignete Sprache*:
Redundanz und Heterogenität der Daten behandeln zu können.

5-Ebenen-Schema-Architektur (Sheth, Larson 1990)



5-Ebenen-Schema-Architektur (Sheth, Larson 1990)

- *Lokales Schema:* Konzeptionelles Schema der Komponente.
- *Komponentenschema:* lokales Schema in einem einheitlichen Datenmodell (kanonisches Datenmodell CDM) reformuliert.
- *Exportschema:* für die Föderation zugelassener Ausschnitt des Komponentenschemas (in CDM).
- *Föderiertes Schema (Globales Schema):* Gesamtheit aller durch die Exportschemata in die Föderation einfließenden Daten, in CDM.
- *Externe Schemata:* Relevante Ausschnitte des föderierten Schemas, in der Regel nicht in CDM.
- *Varianten:*
  - *lose Kopplung:* Benutzer definieren „ihr“ föderiertes Schema bzgl. existierenden Exportschemata.
  - *enge Kopplung:* Eine Föderierungs-DBA verhandelt über Exportschemata und integriert diese in das föderierte Schema.

Vergleich der Referenzarchitekturen

	Architektur		
	Import/Export	Multidatenbank	5–Ebenen–Schema
Kopplungsform	lose Kopplung	lose Kopplung	lose und enge Kopplung
Integration	Benutzer und DBA	Benutzer	Benutzer oder globaler DBA
Zugriff auf Föderation	lokales System	globale Schnittstelle	globales System
globale DBMS-Funktionalität	keine	teilweise	volle angestrebt

Aufgabenstellungen in Multidatenbanksystemen

Architektur

- Komponentendatenbanken müssen selbständig bleiben,
- Lesezugriffe auf globaler Ebene,
- ggf. *zusätzlich* Updates auf globaler Ebene

- Bottom-up: Abbildung (Integration) der Daten der Komponentendatenbanken auf die globale Ebene (Datenmodell)
- Top-Down: Umsetzung der Anfragen/Updates auf globaler Ebene auf die Komponentendatenbanken (Funktionalität)

Integration: Gestaltung der globalen Ebene

- Verteilte Metadaten (Schemata)
- *Heterogenität*
  - der beteiligten Datenbanksysteme (Datenmodell, Anfragesprachen)
  - der Daten (Anwendungsdomäne, Struktur, Semantik)
- Inter-Datenbank-Abhängigkeiten:
  Konsistenzbedingungen,
  falls derselbe Inhalt überlappend in mehreren Datenbanken abgelegt ist
- Konflikte:
  - *Semantische Konflikte:* Es existieren einander entsprechende Klassen, deren Extensionen (Objektmengen) nicht vollständig identisch sind.
  - *Beschreibungskonflikte:* Benennungskonflikte (Homonyme, Synonyme), Wertebereichskonflikte, Maßeinheiten, Konflikte aufgrund von Integritätsbedingungen.
  - *Heterogenitätskonflikte:* unterschiedliche Datenmodelle der Schemata.
  - *Strukturelle Konflikte:* dieselben Zusammenhänge unterschiedlich dargestellt.
  - *Datenkonflikte:* sich widersprechende Daten.

Funktionalität

- Multidatenbank-Anfragesprache
  - für gleichartige Schemata z.B. SchemaSQL
  - beliebige Schemata: Architektur mit Wrapper/Mediator/Anfrageschnittstelle
- Multidatenbank-Updates, globale/verteilte Transaktionen
- Konsistenzerhaltung (Inter-Datenbank-Konsistenz)

Beispiel: Integration von Börseninformationen

Frankfurt::Kurs		
Datum	Name	Wert
3.3.93	sun	150
3.3.93	dc	151
3.3.93	b.u.	160
4.3.93	sun	153
4.3.93	dc	154
4.3.93	b.u.	163

Tokyo::Kurs			
Datum	sun	dc	fuji
3.3.93	150	151	140
4.3.93	153	154	140

New York::sun	
Datum	Wert
3.3.93	150
4.3.93	153

New York::dc	
Datum	Wert
3.3.93	151
4.3.93	154

New York::msoft	
Datum	Wert
3.3.93	148
4.3.93	74

Sydney::3.3.	
Name	Wert
sun	150
dc	151
kiwi	130

Sydney::4.3.	
Name	Wert
sun	153
dc	154
kiwi	135

Verschärfung: Euro vs. Dollar vs. Yen

Integration: Gestaltung der globalen Ebene

- Vergleich der Schemata (Inter-Schema-Korrespondenzen)
- Entwurf eines gemeinsamen Schemas; ggf. (inkrementelle) Anpassung der lokalen Schemata (Entwurf von Views auf die Komponenten)
- Zusammenführung und Restrukturierung der Daten

Schemaintegration

verschiedene Verfahren möglich ...

in dieser Vorlesung:

zusicherungsbasierte Integration (Spaccapietra et al. 1992).

Zusicherungen

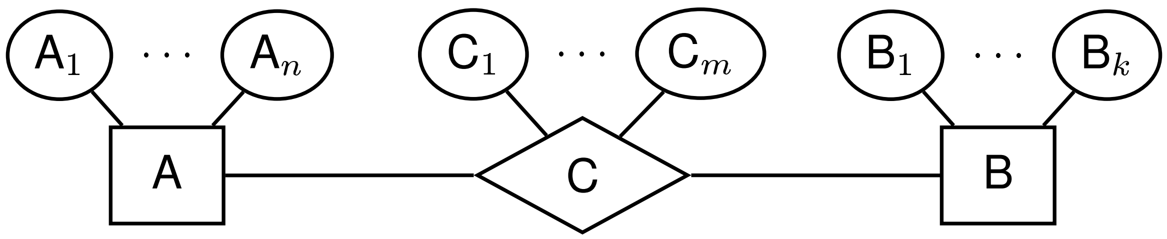
- drücken aus, welche Bestandteile der zu integrierenden Schemata einander entsprechen oder in irgendeiner anderen für die Integration relevanten Beziehung stehen.
- unabhängig von einem konkreten Datenmodell formuliert, sondern
- benutzt ein generisches objektorientiertes Datenmodell

Generisches Datenmodell (GDM)

Konzepte: Objekte, Wertattribute, Referenzattribute.

- Objekte haben eine Objektidentität, der ein Tupel von Attributen zugeordnet ist,
- Für jedes Attribut existieren Min/Max-Angaben zur Eingrenzung der zulässigen Anzahl Werte,
- Attribute können atomar oder komplex sein,
- Komplexe Attribute bestehen wiederum aus einem Tupel von Attributen,
- Atomare Attribute sind entweder Werte oder Referenzen,
- Referenzattribute beschreiben bidirektionale Beziehungen (Links) zwischen Objekten.

Beispiel: ERM nach GDM



GDM-Objekte:

A mit Attributen $A_1,\ldots,A_n$ und einem Referenzattribut A_C,

B mit Attributen $B_1,\ldots,B_k$ und einem Referenzattribut B_C,

C mit Attributen $C_1,\ldots,C_m$ und zwei Referenzattributen C_A und C_B.

Beispiel: Börsenkurse

A = Firma: Name,
B = Börsenplatz: Stadt
C = hat_Kurs: Datum, Wert

Objekt-Korrespondenz-Zusicherungen

sei X_1 aus Schema S_1 und X_2 aus Schema S_2

- $X_1 \equiv X_2$ (*Äquivalenz*):
  X_1 und X_2 repräsentieren immer (in jedem Zustand) dieselbe Menge von Objekten der realen Welt.
- $X_1 \supseteq X_2$ (*Einschluß*):
  Zu jedem Zeitpunkt enthält die Menge von Objekten der realen Welt, die durch X_1 repräsentiert wird, die Menge von Objekten, die durch X_2 repräsentiert wird.
- $X_1 \cap X_2$ (*Überlappung*):
  Zwischen den Mengen von Real–Welt–Objekten, die durch X_1 und X_2 repräsentiert werden, kann es eine Überlappung geben.
- $X_1 \neq X_2$ (*Disjunktheit*): die von X_1 und X_2 repräsentierten Objektmengen sind immer disjunkt, stehen aber in einer Beziehung, die eine Integration erfordert.

Attribut–Zusicherungen

ergänzen (im Fall einer nichtleeren Objektkorrespondenz) die Objektkorrespondenz

- $=$ die Attribute haben immer dieselben Werte.
- $\supseteq$ bei mengenwertige Attributen: Teilmengenbeziehung, sonst: Werte immer gleich oder das auf der rechten Seite stehende Attribut hat einen Nullwert.
- $\cap$ die Attribute sind mengenwertig und ihre Wertemengen können sich schneiden.
- $\neq$ keine der bisherigen Bedingungen gilt – die Attribute stehen aber in einer Beziehung zueinander, die eine Integration erfordert..

Notation:

$X_1 \supseteq X_2$ with corresponding attributes $A_{11} = A_{21},\ A_{12} \cap A_{22}$

Attribut-Zusicherungen

Attribut-Zusicherungen können parametrisiert sein:

- *Bijektionen* auf den Wertebereichen.
  Beispiel: Währungsumrechnungen
- *Aggregierungsfunktionen*, die ein skalares Attribut in Beziehung zu einem mengenwertigen setzen.
  Beispiel: Anzahl Kinder und Menge der Kinder.
- *Kartesisches Produkt*, das ein komplexes Attribut in Beziehung zu mehreren skalaren setzt.
  Beispiel: Adresse zu Stadt, Straße, Hausnummer.

Pfad-Korrespondenz-Zusicherungen

P_1 ein Pfad $X_1 - X_2 - ... - X_n$ in Schema S_1 ($X_i - X_{i+1}$ jeweils eine Beziehung entlang eines Referenzattributes), P_2 ein Pfad $Y_1 - Y_2 - ... - Y_p$ ein Pfad in Schema S_2.

Sei R_1 die Menge der Paare von Objekten aus X_1 und X_n, die über P_1 zueinander in Beziehung stehen, analog für R_2. Dann gibt es die folgenden Pfad–Zusicherungen:

- $P_1 \equiv P_2$ (*Pfadäquivalenz*):
  $R_1 = R_2$ in jedem Datenbankzustand.
- $P_1 \supseteq P_2$ (*Pfadeinschluß*):
  $R_1 \supseteq R_2$ in jedem Datenbankzustand.
- $P_1 \cap P_2$ (*Pfadüberlappung*):
  Es kann Datenbankzustände geben, in denen $R_1 \cap R_2$ nicht leer ist.
- $P_1 \neq P_2$ (*Pfadausschluß*):
  R_1, R_2 sind disjunkt, diese Mengen stehen aber in Beziehung zueinander.

Schemaintegration

Seien $E_1 \equiv E_2$ zwei äquivalente Elemente (Objekttypen oder komplexe Attribute) aus Schemata S_1 und S_2,
E_1 Wertattribute $(A_{11}, ..., A_{1j}, B_1, ..., B_k)$,
E_2 Wertattribute $(A_{21}, ..., A_{2j}, C_1, ..., C_h)$ mit Attributkorrespondenzen
$attcor_1(A_{11}, A_{21}), \ldots, attcor_j(A_{1j}, A_{2j})$
(muß eine Abbildung der Schlüsselattribute beinhalten!).

Objekttyp für das integrierte Schema:

$E :=$ integrate-join $(E_1, E_2, attcor_1(A_{11}, A_{21}), ..., attcor_j(A_{1j,2j}))$

Attribute:

- für jedes Attribut B_i von E_1, zu dem kein Gegenstück von E_2 existiert, ein Attribut B_i' mit gleichem Wertebereich und gleicher Kardinalität,
- für jedes Attribut C_i von E_2, zu dem kein Gegenstück von E_1 existiert, ein Attribut C_i' mit gleichem Wertebereich und gleicher Kardinalität, und
- für jedes Attribut $attcor_i(A_{1i}, A_{2i})$ ein Attribut A_i, das als Integration aus A_{1i} und A_{2i} entsteht.

Schemaintegration

Gemeinsame Attribute:
für jedes Paar $attcor_i(A_{1i}, A_{2i})$ ein Attribut A_i, das als Integration aus A_{1i} und A_{2i} entsteht.

$A_{1i} = A_{2i}$:
: In diesem Fall kann eines der beiden Attribute ausgewählt werden und in das integrierte Schema übernommen werden.

$A_{1i} \supseteq A_{2i}$:
: A_{1i} allgemeiner, wird mit seinem Wertebereich und seiner Kardinalität übernommen.

$A_{1i} \cap A_{2i}$:
: Wertebereich: Vereinigung der Wertebereiche von A_{1i} und A_{2i},
minimale Kardinalität: Maximum der minimalen Kardinalitäten von A_{1i} und A_{2i},
maximale Kardinalität: Summe von A_{1i} und A_{2i}.

$A_{1i} \neq A_{2i}$:
: Wertebereich: Vereinigung der Wertebereiche von A_{1i} und A_{2i},
minimale und maximale Kardinalität: jeweils Summe der minimalen bzw. maximalen Kardinalitäten von A_{1i} und A_{2i}.

Integrationsregeln

(nur für Äquivalenzzusicherungen)

- Im Falle von strukturellen Konflikten enthält das integrierte Schema die allgemeinere Struktur.

Integrationsregel 1

Objekte ohne Gegenstück

Für jedes Element $X1$ aus Schema $S1$, zu dem es kein Gegenstück im Schema $S2$ gibt, wird ein Element X im integrierten Schema (GDM) erzeugt. X hat dabei denselben Typ wie $X1$.

Für jeden Link $X1 - Y1$ in Schema $S1$, zu dem es kein Gegenstück im Schema $S2$ gibt, wird entsprechend ein Link in dem integrierten Schema (GDM) zwischen X und Y eingefügt.

Integrationsregel 2

Äquivalente Objekte

Seien E_1 und E_2 Objekttypen in Schema S_1 bzw. S_2 mit Wertattributen $(A_{11}, ..., A_{1j}, B_1, ..., B_k)$ bzw. $(A_{11}, ..., A_{1j}, C_1, ..., C_h)$ und

$E_1 \equiv E_2$ with corresponding attributes:
$attcor_1(A_{11}, A_{21}), ..., attcor_j(A_{1j}, A_{2j})$.

Dann wird in das integrierte Schema ein Objekttyp

$E :=$ integrate-join $(E_1, E_2, attcor_1(A_{11}, A_{21}), ..., attcor_j(A_{1j,2j}))$

aufgenommen.

Integrationsregel 3

Elementare Links zwischen äquivalenten Objekten

Seien $A_1 \equiv A_2$ zu A integriert, $B_1 \equiv B_2$ zu B integriert und $A_1 - B_1 \equiv A_2 - B_2$.

Dann enthält das integrierte Schema ein Link $A - B$.

Integrationsregel 4

Pfade im integrierten Modell

Seien $A_1, B_1, ..., D_1$ Elemente in S_1, $A_2, B_2, ..., D_2$ Elemente in Schema S_2 mit $A_1 \equiv A_2$ und $D_1 \equiv D_2$.

Wenn A und D die integrierten Elemente zu A_1 und A_2 bzw. D_1 und D_2 sind, dann gilt:

- Die Zusicherung $A_1 - D_1 \equiv A_2 - B_2 - ... - D_2$ zwischen einem elementaren Link und einem Pfad erzeugt im integrierten Schema den Pfad $A - B_2' - ... - D$ (wobei B_2' das integrierte Element zu B_2 ist).
- Die Zusicherung $A_1 - B_1 - ... - D_1 \equiv A_2 - B_2 - ... - D_2$ zwischen zwei Pfaden erzeugt in dem integrierten Schema zwei Pfade $A - B_1' - ... - D$ und $A - B_2' - ... - D$ sowie eine Integritätsbedingung daß diese beiden Pfade immer die gleichen Instanz-Paare von A und D zueinander in Beziehung setzen.

Integrationsregel 5

Objekttypen und komplexe Attribute

X_1 ein Objekttyp aus Schema S_1 mit Wertattributen $(A_{11}, ..., A_{1j}, B_1, ..., B_1)$,

X_2 ein komplexes Attribut eines Elementes E_2 (nach obigen Regeln in einen Objekttyp E integriert) aus Schema S_2 mit den Wertattributen $(A_{21}, ..., A_{2j}, C_1, ..., C_h)$ als Komponenten.

$X_1 \equiv X_2$ mit Attributkorrespondenzen $attcor_1(A_{11}, A_{21}), ..., attcor_j(A_{1j}, A_{2j})$

Integration:
$X =$ integrate-join $(X_1, X_2, attcor_1(A_{11}, A_{21}), ..., attcor_j(A_{1j,2j}))$

Das Attribut X_2 von E_2 wird in ein Referenzattribut X_2' (auf X) von E transformiert. Die Kardinalitäten von X_2' sind dieselben wie die von X_2.

Aufgabe: Integration von Objekttypen

Definieren sie auf Basis von integrate-join und Regel 5 einen Operator, der Korrespondenzen komplexer Attribute direkt verarbeitet

Wenden Sie Ihre Lösung auf die Objekttypen

- Person: Vorname, Nachname, Adresse und
- Person': Name, Strasse, Hausnummer, Ort, PLZ

an, indem Sie geeignete naheliegende Attribut-Korrespondenzen verwenden.

Lösung

Seien $E = (A_1, \ldots, A_n, B_1, \ldots, B_k)$ und
$E' = (A'_1, \ldots, A'_k, B'_1, \ldots, B'_l)$ so daß für
$1 = i_0 \leq i_1 < i_2 < \ldots < i_m = n$ und
$1 = j_0 \leq j_1 < i_2 < \ldots < i_m = n'$ die Attribut-Äquivalenzen
$\mathsf{attcor}((A_{i_{l-1}+1}, \ldots, A_{i_l}), (A'_{j_{l-1}+1}, \ldots, A'_{j_l}))$ gelten. Dann ist

$$\mathsf{integrate\text{-}join}^*(E, E', \mathsf{attcor}((A_1, \ldots, A_{i_1}), (A'_1, \ldots, A'_{j_1})), \ldots \\ \mathsf{attcor}((A_{i_{m-1}+1}, \ldots, A_{i_m}), (A'_{j_{m-1}+1}, \ldots, A'_{j_m})))$$

definiert wie folgt:

Für jedes (komplexe) Attribut $(A_{i_{l-1}+1}, \ldots, A_{i_l})$ wird ein Objekttyp in das integrierte Schema aufgenommen. Dieser erhält ein Link als Referenzattribut an E und E'; Wertebereich und Kardinalitäten werden wie in der Vorlesung definiert.

Lösung

Neue komplexe Objekttypen: NameT $=$ (Vorname,Nachname), AdresseT $=$ (Strasse,Hausnummer,Ort,PLZ)

Attributkorrespondenzen:

attcor((Vorname,Nachname),Name) :
$$(V, N) \equiv concatenate(V, N)$$
attcor(Adresse,(Strasse,Hausnummer,Ort,PLZ)) :
$$concatenate(S, N, \text{“,”}, P, O) \equiv (S, N, P, O).$$

Damit ist Person $\equiv$ Person'

mit (Vorname,Nachname) $\equiv$ Name und

Adresse $\equiv$ (Strasse,Hausnummer,Ort,PLZ)

Das Ergebnis ist z.B. Person*: Name, Adresse, wobei Name und Adresse als komplexe Objekttypen (mit den entsprechenden Attributen) in dem integrierten Schema enthalten sind.

Aufgabe: Pfadkorrespondenzen

Für welche der in der Vorlesung genannten Pfadkorrespondenzen $\otimes \in \{\equiv, \supseteq, \cap, \neq\}$ gilt

$$\begin{aligned} \text{Wenn} \quad & (X_1 - \ldots - X_m) \otimes (Y_1 - \ldots - Y_k) \\ \text{und} \quad & (X_m - \ldots - X_n) \otimes (Y_k - \ldots - Y_l)\ , \\ \text{dann} \quad & (X_1 - \ldots - X_n) \otimes (Y_1 - \ldots - Y_l)\ \ ? \end{aligned}$$

Lösung

Beweis für Pfadeinschluß:

Sei $(X_1 - \ldots - X_m) \subseteq (Y_1 - \ldots - Y_k)$ und $(X_m - \ldots - X_n) \subseteq (Y_k - \ldots - Y_l)$, d.h. für zwei Objekte o_1 und o_2 so daß ein Pfad wie durch $(X_1 - \ldots - X_m)$ gegeben von der Repräsentation von o_1 in Schema 1 zu der Repräentation von o_2 in Schema 1 existiert, existiert ein Pfad wie durch $(Y_1 - \ldots - Y_k)$ gegeben von der Repräsentation von o_1 in Schema 2 zu der Repräsentation von o_2 in Schema 2:

$$([o_1]_{S1}, [o_2]_{S1}) \in (X_1 - \ldots - X_m) \Rightarrow ([o_1]_{S2}, [o_2]_{S2}) \in (Y_1 - \ldots - Y_k)\,;$$

analog für

$$([o_2]_{S1}, [o_3]_{S1}) \in (X_m - \ldots - X_n) \Rightarrow ([o_2]_{S2}, [o_3]_{S2}) \in (Y_k - \ldots - Y_l)\,.$$

Sei nun o_1 und o_3 so daß $([o_1]_{S1}, [o_3]_{S1}) \in (X_1 - \ldots - X_n)$ mit o_2 so daß $([o_1]_{S1}, [o_2]_{S1}) \in (X_1 - \ldots - X_m)$ und $([o_2]_{S1}, [o_3]_{S1}) \in (X_m - \ldots - X_n)$.
Dann ist $([o_1]_{S2}, [o_2]_{S2}) \in (Y_1 - \ldots - Y_k)$ und $([o_2]_{S2}, [o_3]_{S2}) \in (Y_k - \ldots - Y_l)$, womit $([o_1]_{S2}, [o_3]_{S2}) \in (Y_1 - \ldots - Y_l)$ ist.

Aus $\subseteq$ und $\supseteq$ folgt die Aussage für Pfadäquivalenz.

Lösung

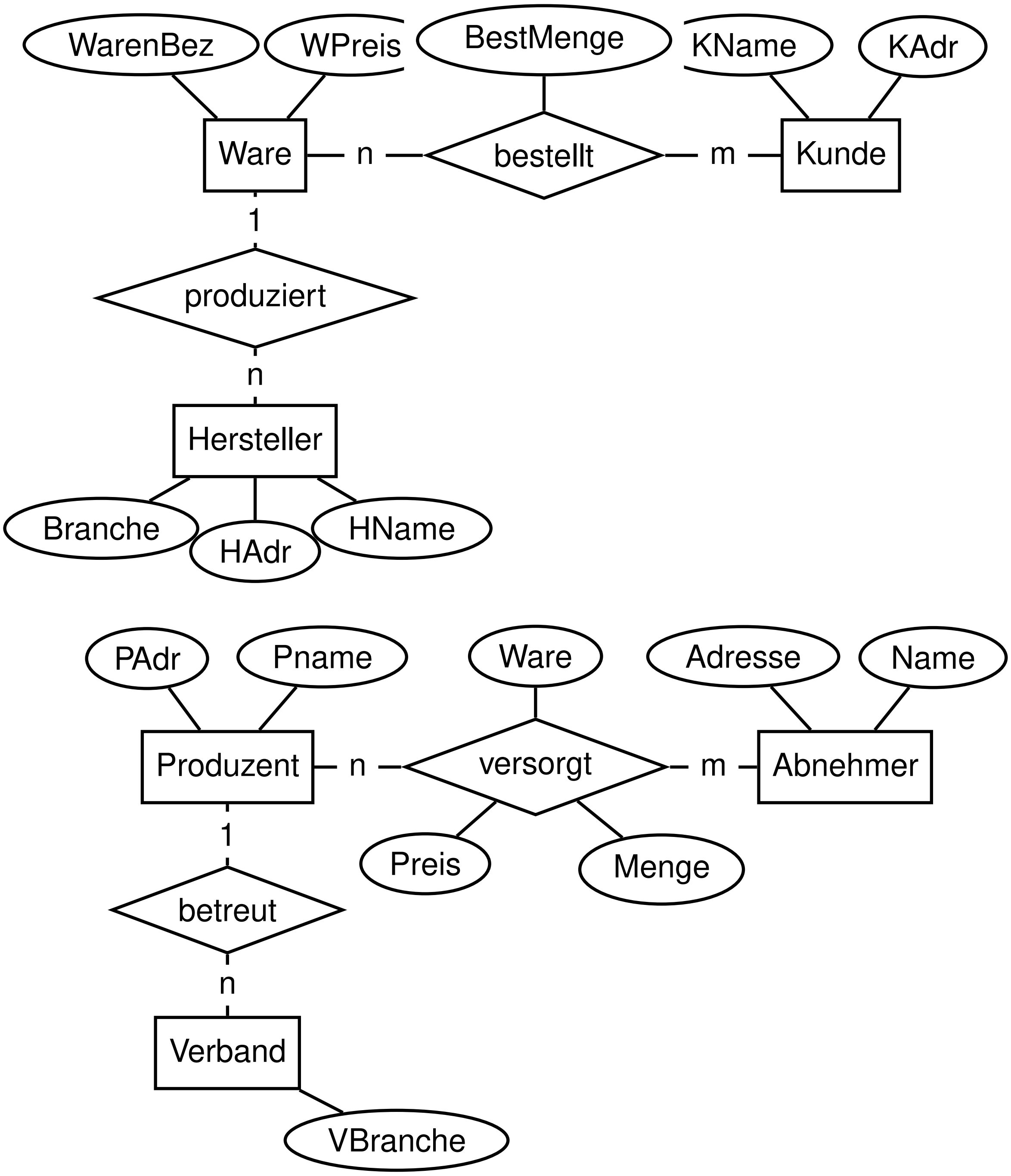
Für Pfadüberlappung gilt es ebenfalls. Insbesondere bleibt die Aussage gültig, wenn die Schnittmenge eine Funktion der beteiligten Objektmengen ist.

Die Eigenschaft gilt i.a. nicht für Pfadausschluß (die Eigenschaft gilt für den Teil “stehen in Beziehung zueinander” des Pfadausschlusses, aber nicht notwendigerweise für den Teil “Menge der Paare ist disjunkt”).

Man betrachte z.B. $(x_1, x_2) \neq (x_1 - \ldots - x_{n-1})$ und $(x_{n-1}, x_n) \neq (x_2 - \ldots - x_n)$, dennoch ist trivialerweise $(x_1 - \ldots - x_n) \equiv (x_1 - \ldots - x_n)$.

Beispiel

Die folgenden ER-Schemata sollen integriert werden:



Beispiel: Korrespondenz-Zusicherungen

1. KUNDE $\equiv$ ABNEHMER
   with corresponding attributes:
   KName $=$ Name
   KAdr $=$ Adresse
2. HERSTELLER $\equiv$ PRODUZENT
   with corresponding attributes:
   HName $=$ PName
   HAdr $=$ PAdr
3. WARE $\equiv$ versorgt
   with corresponding attributes:
   WarenBez $=$ Ware
   WPreis $=$ Preis
4. HERSTELLER–produziert–WARE–WarenBez $\equiv$
   PRODUZENT–versorgt–Ware
5. WARE–bestellt–KUNDEN $\equiv$ versorgt–ABNEHMER

Beispiel: Anwendung der Integrationsregeln

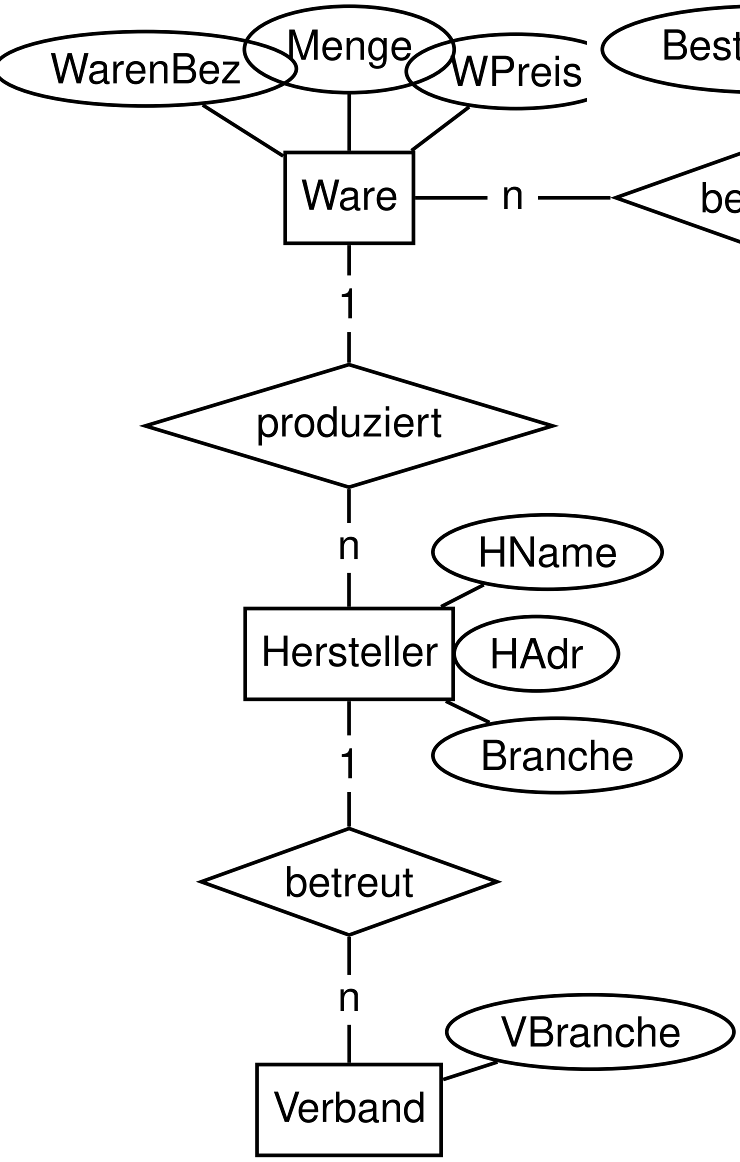
1. *Regel 2*: Aus S1.KUNDEN und S2.ABNEHMER wird im integrierten Schema der Entitytyp KUNDEN mit den Attributen KName und KAdr erzeugt.
2. *Regel 2*: Aus S1.HERSTELLER und S1.HERSTELLER wird im integrierten Schema HERSTELLER mit den Attributen HName, HAdr und Branche erzeugt.
3. *Regel 2*: Aus S1.WAREN und S2.versorgt wird ein Entitytyp WAREN im integrierten Schema mit den Attributen WarenBez, WPreis und Menge erzeugt.

Anwendung der Integrationsregeln:

Regel 1: Übernahme von Schemaelementen ohne Gegenstück.

- der Beziehungstyp bestellt mit seinem Attribut BestMenge aus Schema S1
- der Beziehungstyp produziert aus Schema S1
- der Entitytyp VERBAND mit seinem Attribut VBranche aus Schema S2
- der Link (die Verbindung) zwischen bestellt und KUNDE aus Schema S1
- der Link zwischen bestellt und WARE aus Schema S1
- der Link zwischen produziert und WARE aus Schema S1
- der Link zwischen produziert und HERSTELLER aus Schema S1
- der Link zwischen betreut und PRODUZENT (im integrierten Schema dann HERSTELLER) aus Schema S2
- der Link zwischen betreut und VERBAND aus Schema S2

Beispiel: Integriertes Entity-Relationship-Schema



Beispiel

- Welche Daten enthalten die Schemata S1, S2, sowie das integrierte Schema für den folgenden Ausschnitt einer Anwendung: Eine Firma X liefert n Türschlösser an eine andere Firma F zum Preis P, und m Türschlösser an eine Firma G zum Preis $Q \neq P$.
- Geben Sie alle Attribute der Relation “versorgt” in Schema S2 an, kennzeichnen Sie Schlüsselattribute.
- Betrachten Sie die Views (HName, WarenBez, Preis, BestMenge, KName) (S1) und (PName, Ware, Preis, Menge, Abnehmer.Name) (S2). Welche funktionalen Abhängigkeiten bestehen dort ?

Beispiel

- S1: Hersteller: (X,_,_)
  Kunde: (F,_,_), (G,_,_)
  Ware: (T,P), (T,Q)
  produziert: (X,T,P), (X,T,Q)
  bestellt: (T,P,n,F), (T,Q,m,G)

  S2: Produzent: (X,_,_)
  Abnehmer: (F,_,_), (G,_,_)
  versorgt: (X,T,P,n,F), (X,T,Q,m,G)

  I.S: Hersteller: (X,_,_)
  Kunde: (F,_,_), (G,_,_)
  Ware: (T,n'>n,P), (T,m'>m,Q)
  bestellt: (T,P,n,F), (T,Q,m,G)

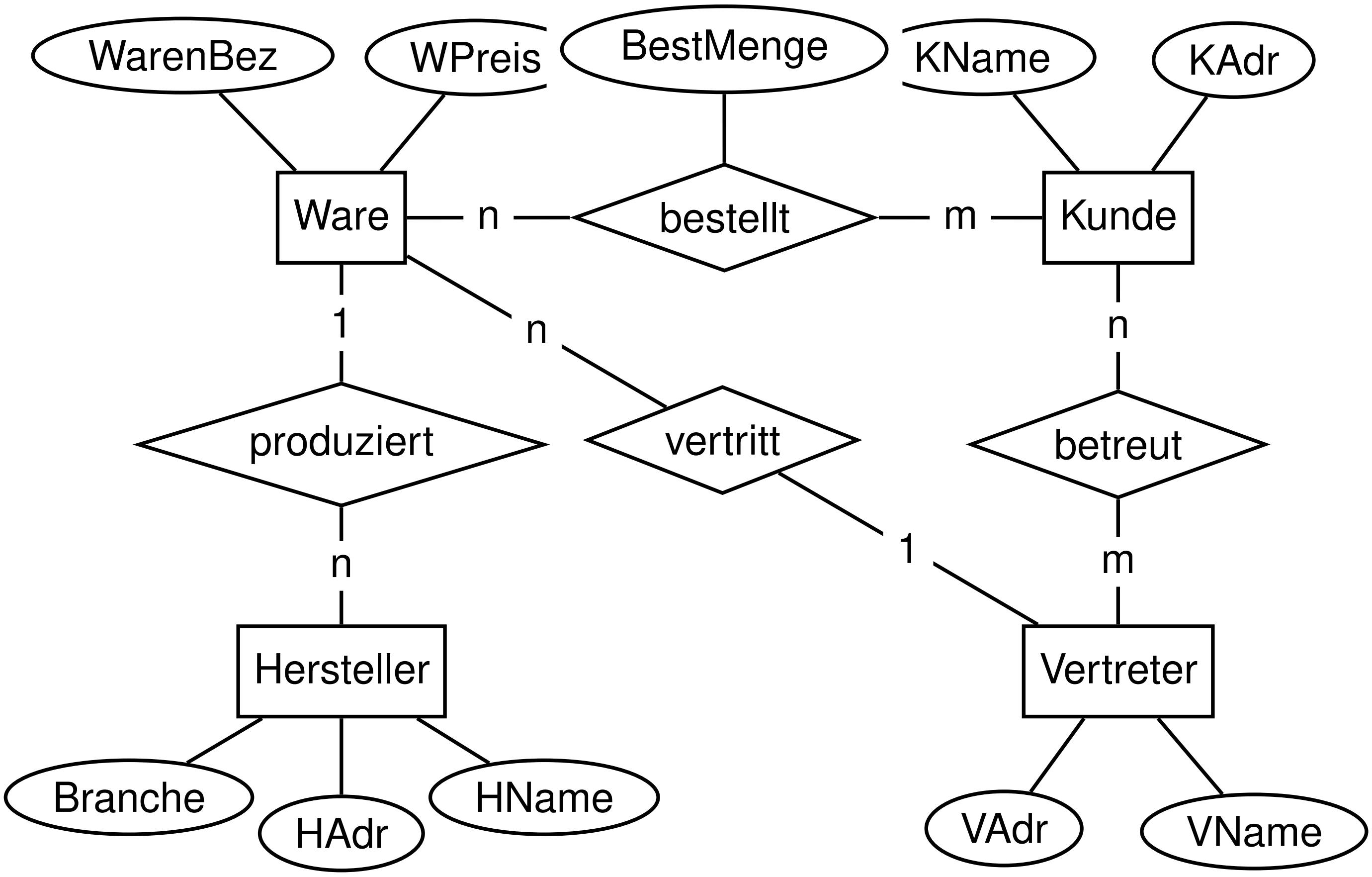
- Modellierung wegen dem zusätzlichen Attribut “Menge” bei “Ware” nicht glücklich
  (Jahresproduktion? Gesamtproduktion? Menge einer Bestellung?).

Beispiel

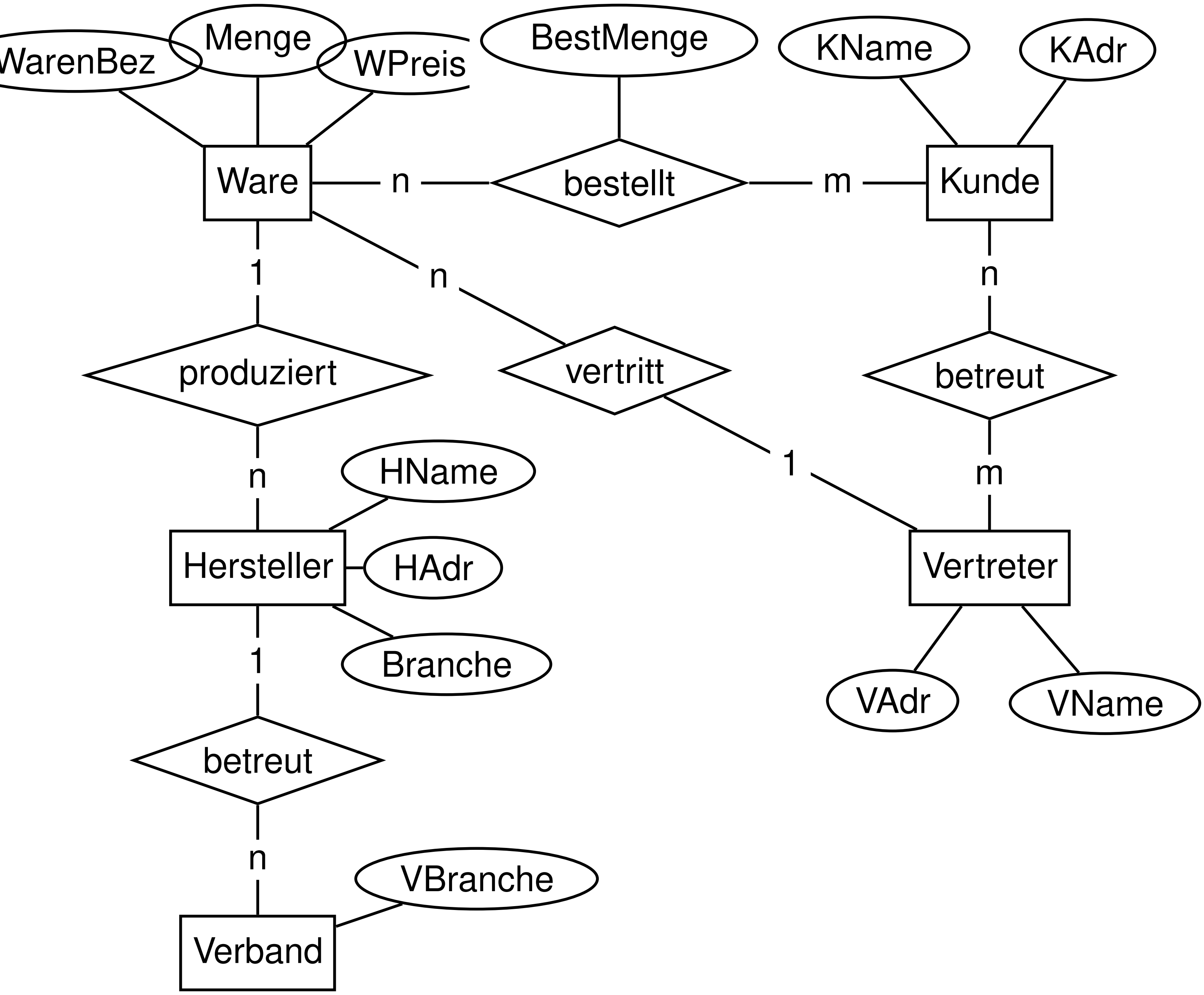
- Relation “versorgt”: PName (von Produzent), Ware, Preis, Menge, Name (von Abnehmer). Schlüsselattribute sind mindestens PName, Ware, Name.
- Funktionale Abhängigkeiten:
  S1: WarenBez $\Rightarrow$ HName ist die wichtigste; ein Äquivalent in S2 existiert *nicht* (n,m-Beziehung). Damit ist eine Integration in ein Schema, das diese funktionale Abhängigkeit (per Kardinalität (Waren,produziert)) fordert, nur möglich, wenn diese funktionale Abhängigkeit in der zweiten Datenbasis “zufällig” erfüllt ist.
  Im Prinzip liegt hier eine Dreierbeziehung Hersteller-Ware-Verbraucher vor, die genau dann verlustfrei zerlegt werden kann, wenn man eine funktionale Abhängigkeit von einem Attribut findet (vgl. S1)

Beispiel

Betrachten Sie die folgende Ergänzung zu dem Schema S1 und führen sie eine Integration mit S2 durch. Welche Probleme treten jetzt auf ?



Beispiel: Ergänztes Modell



Beispiel: Problem

- Schema S1: “Preis” ist einer bestimmten Ware zugeordnet
- Schema S2: “Preis” ist einer bestimmten “versorgt”-Beziehung zugeordnet
- Zusicherung

  $$\mathsf{Waren} \equiv \mathsf{versorgt}: \quad \mathsf{WarenBez} \equiv \mathsf{Ware}, \quad \mathsf{WPreis} \equiv \mathsf{Preis}$$

  nicht ganz korrekt.
- integriertes Schema: Attribut “Preis” ist einer Ware zugeordnet
- unterschiedliche Preisvereinbarungen mit verschiedenen Bestellern möglich?
  Mehrere “Ware”-Instanzen desselben Produkts
- Vertreter soll nur eine Ware/Produkt vertreten
  $\Rightarrow$ Modellierungsproblem

Beispiel: Problem

In S2 enthält der Beziehungstyp “versorgt” mit seinen Attributen einen *versteckten Entitytyp*, der betrachtet werden müßte:

$$\mathsf{Waren.Waren} - \mathsf{bestellt} - \mathsf{Menge} \equiv \mathsf{versorgt.Menge}$$

$\Rightarrow$ *Normalisierung* vor der Integration.

In einer korrekten Modellierung muß das Attribut “Preis” der Beziehung zwischen Ware und Kunde zugeordnet sein.

- Die Regeln zur Schemaintegration geben nur eine Vorgehensweise vor, die in Einzelfällen validiert werden muß.

Globale Integrität

- Mehr als nur die Summe aller lokalen Integritätsbegriffe
  - Integration lokaler Integritätsbedingungen (z.T. unterschiedlich streng in den beteiligten DBn),
  - z.T. aus Schemaintegration entstanden (vgl. Integrationsregel 4),
  - Integritätsbedingungen durch Datenmodellwechsel (z.B. Subklassenbedingungen in relationales Modell umsetzen)
  - globale Schlüsselbedingungen
  - neu dazukommende Inter-Datenbank-Bedingungen

Integrität in den Referenzarchitekturen

- Import-Export-Schema: kein globales Schema
  jedes Komponentensystem muß über relevante globale Integritätsbedingungen informiert sein und sie berücksichtigen.
- Multidatenbank: kein globales Schema
  zusätzliche Abhängigkeitsschemata unter Bezugnahme auf die lokalen Schemata vorgesehen.
- 5-Ebenen-Schema: explizit ein globales Schema vorhanden.
  auf dieser Ebene können globale Integritätsbedingungen verwaltet werden.

Integritätserhaltung

Was tun bei einer globalen Integritätsverletzung durch ein globales Update oder ein Update an einer Komponentendatenbank?
(Probleme können also nicht nur durch globale, sondern auch durch lokale Updates entstehen).

- Autonomie eingeschränkt
- ablehnen?
- integritätserhaltende Updates auf anderen Komponentendatenbanken?

Einfügen/Löschen von Objekten

- Semantisch äquivalente Extensionen:
  - Globales Einfügen: Transformation in lokale Objekte und lokale Abspeicherung. Lokale IB beachten. GGf. global zurückweisen.
  - Lokales Einfügen: Transformation in globales Objekt, weiter wie oben.
  - Löschen: analoge Problematik.
- Semantisch überlappende Extensionen:
  - Globales Einfügen: in welche Komponentendatenbank(en) muß/kann/soll ein lokales Objekt eingefügt werden?
  - Lokales Einfügen: Ist das Objekt global schon vorhanden? Muß das Objekt auch in andere Komponentendatenbanken eingefügt werden?
  - Globales Löschen: In welchen Komponentendatenbanken muß ebenfalls gelöscht werden?
  - Lokales Löschen: Muß das Objekt auch global oder in anderen Komponentendatenbanken gelöscht (oder eingefügt (Migration)) werden?

Einfügen/Löschen von Objekten

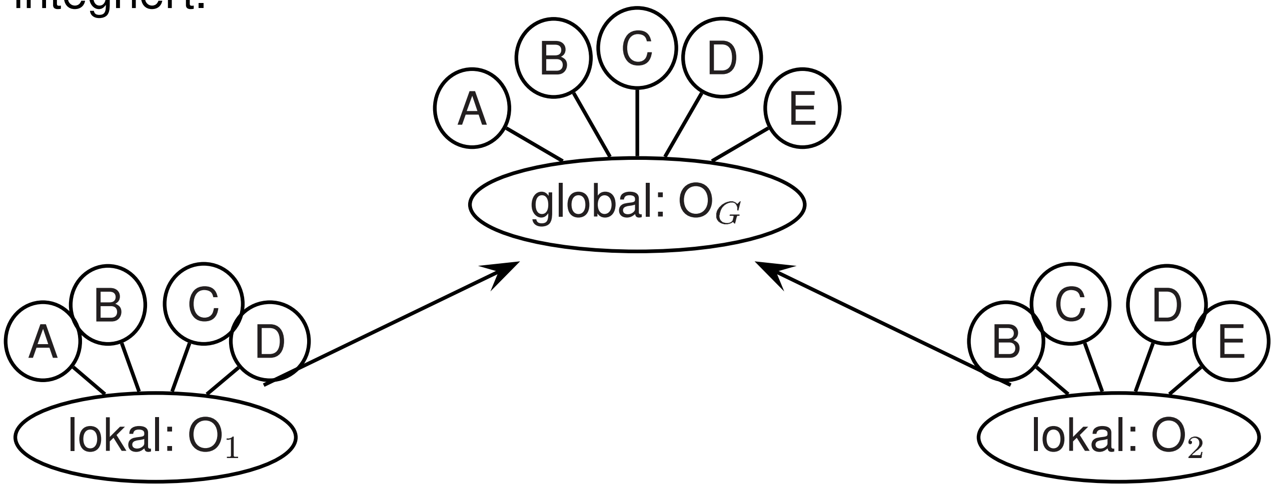
- Disjunkte, aber zusammengehörige Extensionen:
  - Globales Einfügen: in welche Komponentendatenbank muß/kann/soll ein lokales Objekt eingefügt werden?
  - Lokales Einfügen: Ist das Objekt schon in einer anderen Komponentendatenbank vorhanden? Global einfügen.
  - Globales Löschen: Lokale Löschungen.
  - Lokales Löschen: Global auch löschen oder woanders einfügen?

- oft mehrere sinnvolle Möglichkeiten
- Anwendungsabhängig

$\Rightarrow$ Muß in den Anwendungsprogrammen explizit angegeben werden.

Updates an Objekten

Situation: Objekte aus unterschiedlichen Komponentendatenbanken sind zu einem globalen Objekt integriert.



- Globales Update: auf Komponentendatenbanken umsetzen.
- Lokales Update: auf globales Objekt umsetzen.
  Von dort auf andere Komponenten weitergeben (i.a. ja)?

Konsistenzsicherungsmechanismen

- aktuelle DBMS: Trigger
- aktive DBS: ECA-Regeln (Event-Condition-Action)
- Problem: vorhandene lokale Mechanismen von Föderation i.a. nicht zugreifbar.

Protokollbasierte Ansätze

Protokolle zur Kommunikation zwischen Komponentendatenbanken (existieren bereits für verteilte DB)
Jede Komponentendatenbank besitzt einen *Constraint Manager*, der alle relevanten globalen Integritätsbedingungen kennt und Anfragen an andere Komponentendatenbanken stellen kann.

- Einschränkung der Autonomie
- Altsysteme besitzen im Allgemeinen keinen Constraint Manager
- Heterogene Komponentendatenbanksysteme?

Konsistenzsicherung durch Architekturerweiterung

Annahme:

- Komponentensysteme haben keine Information über ihre Teilnahme an Föderation und über andere Komponenten.

Einsatz eines *Wrappers* zwischen Föderierungsebene und Komponente:

- bietet Schnittstelle für lokale Anwendungen
  (ist damit über alle Veränderungen informiert)
- verfügt über Informationen, welche Änderungen potentiell Integritätsverletzungen auslösen können,
- leitet entsprechende Operationen an Föderierungsebene weiter.

– mögliche Probleme bei aktiven oder objektorientierten DBn:
  Methodenanwendungen intern im System → Wrapper nicht informiert

– Effizienzverlust durch Zusatzschich

+ Wrapper bietet Möglichkeiten, weitere Funktionalität zu realisieren.

Multidatenbanksprachen

Interoperabilität: Fähigkeit, heterogene Informationen der Komponentendatenbanken in einheitlicher Weise zu nutzen.

Anforderungen an eine Anfragesprache zur Gewährleistung von Interoperabilität:

- Anfragesprache muß auf Anfragesprachen der beteiligten Datenbanken umsetzbar sein,
- einheitliches Datenmodell der konzeptuellen Schemata, ggf. Abbildung der lokalen Schemata darauf.
- Relationale Datenbanken: Abwärtskompatibilität mit SQL,
- Implementierbarkeit ohne Verletzung der Autonomie der Komponenten.

SchemaSQL

(Lakshmanan et al. 1996)
url: http://www.cs.concordia.ca/ faculty/laks/papers.html

SchemaSQL erweitert SQL:

- Kombination von Relationen und Attributen unterschiedlicher Datenbanken.
- uniforme Behandlung von Daten und Metadaten (durch SchemaSQL-Variablen).
- Mögliche Definitionsbereiche von Variablen sind die Namen der Komponenten der Föderation, Namen der Relationen einer Datenbank, Namen der Attribute einer Relation, Tupel einer Relation, und Werte einer Spalte einer Relation.
- Zusätzlich zu “vertikalen” Aggregierungen über Spalten sind “horizontale” Aggregierungen über Relationen oder Tabellen möglich.

Beispiel

univ-A

salInfo

category	dept	salFloor
Prof	CS	65,000
Assoc Prof	CS	50,000
Prof	Math	60,000
Assoc Prof	Math	55,000

univ-C

CS

category	salFloor
Prof	60,000
Assoc Prof	55,000

Math

category	salFloor
Prof	70,000
Assoc Prof	60,000

univ-B

salInfo

category	CS	Math
Prof	55,000	65,000
Assoc Prof	50,000	55,000

univ-D

salInfo

dept	Prof	Assoc Prof
CS	75,000	60,000
Math	60,000	45,000

Deklaration von Variablen

... wie von SQL bekannt in der FROM-Zeile:

FROM <range> <var>

SQL: FROM <table> <var>

SELECT cty.name, cty.population
FROM City cty

<range> $\in \{\rightarrow,$ db $\rightarrow,$ db :: rel, db :: rel $\rightarrow,$ db :: rel.attr$\}$.

- $\rightarrow$ benennt die Menge der Datenbanknamen der Föderation.
- $db \rightarrow$ benennt die Menge der Relationsnamen der Datenbank db.
- $db :: rel$ benennt die Menge der Tupel der Relation rel der Datenbank db [wie in SQL].
- $db :: rel \rightarrow$ benennt die Menge der Attributnamen des Schemas der Relation rel der Datenbank db.
- $db :: rel.attr$ benennt die Menge der Werte in der Spalte des Attributs $attr$ der Relation rel der Datenbank db.
- SQL: range-Spezifikation muß eine Konstante sein
- SchemaSQL: Aufeinander aufbauende Deklarationen sind erlaubt!
  $\Rightarrow$ Joins nicht mehr symmetrisch (vgl. OQL).

Deklaration von Variablen

- $\rightarrow$ benennt die Menge der Datenbanknamen der Föderation.
  "db-name"-Variable

  SELECT dbname FROM $\rightarrow$ dbname

dbname
"univ-a"
"univ-b"
"univ-c"
"univ-d"

Deklaration von Variablen

- $db \rightarrow$ benennt die Menge der Relationsnamen der Datenbank db.

  “rel-name”-Variable

  SELECT relname FROM univ-C $\rightarrow$ relname

relname
“CS”
“math”

- Geschachtelte Deklaration:
  Zweite Variable ist von der ersten abhängig:

  SELECT dbname, relname
  FROM $\rightarrow$ dbname, dbname $\rightarrow$ relname

dbname	relname
“univ-A”	“SalInfo”
“univ-B”	“SalInfo”
“univ-C”	“CS”
“univ-C”	“math”
“univ-D”	“SalInfo”

  ... kann in symmetrische Anfrage mit Hilfe von Meta-Tabellen umgesetzt werden.

Deklaration von Variablen

- $db :: rel$ benennt die Menge der Tupel der Relation rel der Datenbank db.

  tupel-Variable

  SELECT tupel.category, tupel.salFloor FROM univ-B::CS tupel

category	salFloor
"Prof"	60000
"AssocProf"	55000

  SELECT dept, tupel.category, tupel.salFloor
  FROM univ-B $\rightarrow$ dept, univ-B::dept tupel

dept	category	salFloor
"CS"	"Prof"	60000
"CS"	"AssocProf"	55000
"Math"	"Prof"	70000
"Math"	"AssocProf"	60000

Deklaration von Variablen

- $db :: rel \rightarrow$ benennt die Menge der Attributnamen des Schemas der Relation rel der Datenbank db.
  “attr-name”-Variable

  SELECT attrname
  FROM univ-C::CS $\rightarrow$ attrname

attrname
“category”
“SalFloor”

  SELECT attrname, univ-C::CS.attrname
  FROM univ-C::CS $\rightarrow$ attrname

“category”	“Prof”
“category”	“AssocProf”
“SalFloor”	60000
“SalFloor”	55000

- `SELECT C`: *Bezeichner* des Attributes,
  `SELECT T.C`: jeweiliger Attribut*wert* des betrachteten Tupels.

Deklaration von Variablen

- $db :: rel.attr$ benennt die Menge der Werte in der Spalte des Attributs $attr$ der Relation rel der Datenbank db domain-Variable.

  SELECT x
  FROM univ-A::salInfo.category cat

cat
hline "Prof"
"AssocProf"

Anfragen

Alle Fachbereiche von Univ-A, die ihren Professoren ein höheres Gehalt zahlen, als die entsprechenden Fachbereiche in Univ-B:

```
select A.dept
  – alle Variablen unabhängig voneinander
from   univ-A::salInfo A, univ-B::salInfo B,
       univ-B::SalInfo-> AttB
where  AttB <> = “category” and
       A.dept = AttB and
       A.category = “Prof” and
       B.category = “Prof” and
       A.salFloor > B.AttB.
```

Dasselbe für C/D:

```
select RelC
  – C abhängig von RelC
from   univ-C-> RelC, univ-C::RelC C,
       univ-D::salInfo D
where  RelC = D.dept and
       C.category = “Prof” and
       C.salFloor > D.Prof
```

Horizontale Aggregation

Wie in SQL kann man über eine Variable aggregieren.

⇒ hier auch *horizontale* und *blockweise* Aggregation möglich.

Durchschnittliches Gehalt für jede Mitarbeitergruppe über alle Fachbereiche von univ-C:

select T.category, avg(T.D)
from univ-B::salInfo→D,   univ-B::salInfo T
where D <> “category”
group by T.category

- Werte für D bestimmen,
- kartesisches Produkt mit univ-B::salInfo T bilden

D	category	CS	Math	T.D
category	Prof	55,000	65,000	55,000
CS	Prof	55,000	65,000	55,000
math	Prof	55,000	65,000	65,000
category	Assoc Prof	50,000	55,000	50,000
CS	Assoc Prof	50,000	55,000	50,000
math	Assoc Prof	50,000	55,000	55,000

- Spalte T.D mit einbeziehen
- auswerten, gruppieren, aggregieren

Aggregation

Durchschnittliches Gehalt für jede Mitarbeitergruppe über alle Fachbereiche von univ-C:

select T.category, avg(T.salFloor)
from univ-C→D, univ-C::D T
group by T.category

- Werte für D bestimmen,
- Join mit Tupelvariable D T

D	category	salFloor
CS	Prof	60,000
CS	Assoc Prof	55,000
math	Prof	70,000
math	Assoc Prof	60,000

- gruppieren
- aggregieren

Restrukturierung

... wie üblich:

```
create view
BtoA::salInfo(category, dept, salFloor) as
    select T.category, D, T.D
    from univ-B::salInfo→D,
            univ-B::salInfo T
    where D <> `category'
```

erzeugt eine virtuelle Datenbank BtoA mit einer virtuellen Relation salInfo in demselben Format wie A::salInfo.

Restrukturierung

A to B: Breite der auszugebenden Tabelle hängt von der Anzahl der Departements ab.
$\Rightarrow$ Dynamisches Ausgabeschema

create view AtoB::salInfo(category,D) as
select A.category, A.salFloor
from univ-A::salInfo A, A.dept D

Ergebnis der FROM-Zeile:

A.category	A.salFloor	A.dept D
Prof	65,000	CS
Assoc Prof	50,000	CS
Prof	60,000	Math
Assoc Prof	55,000	Math

Many-To-One-Mapping in ein Schema der Form

$$\text{salInfo(category,dept}_1, \ldots, \text{dept}_n).$$

AtoB::salInfo		
category	CS	Math
Prof	65,000	60,000
Assoc Prof	50,000	55,000

Aufgabe

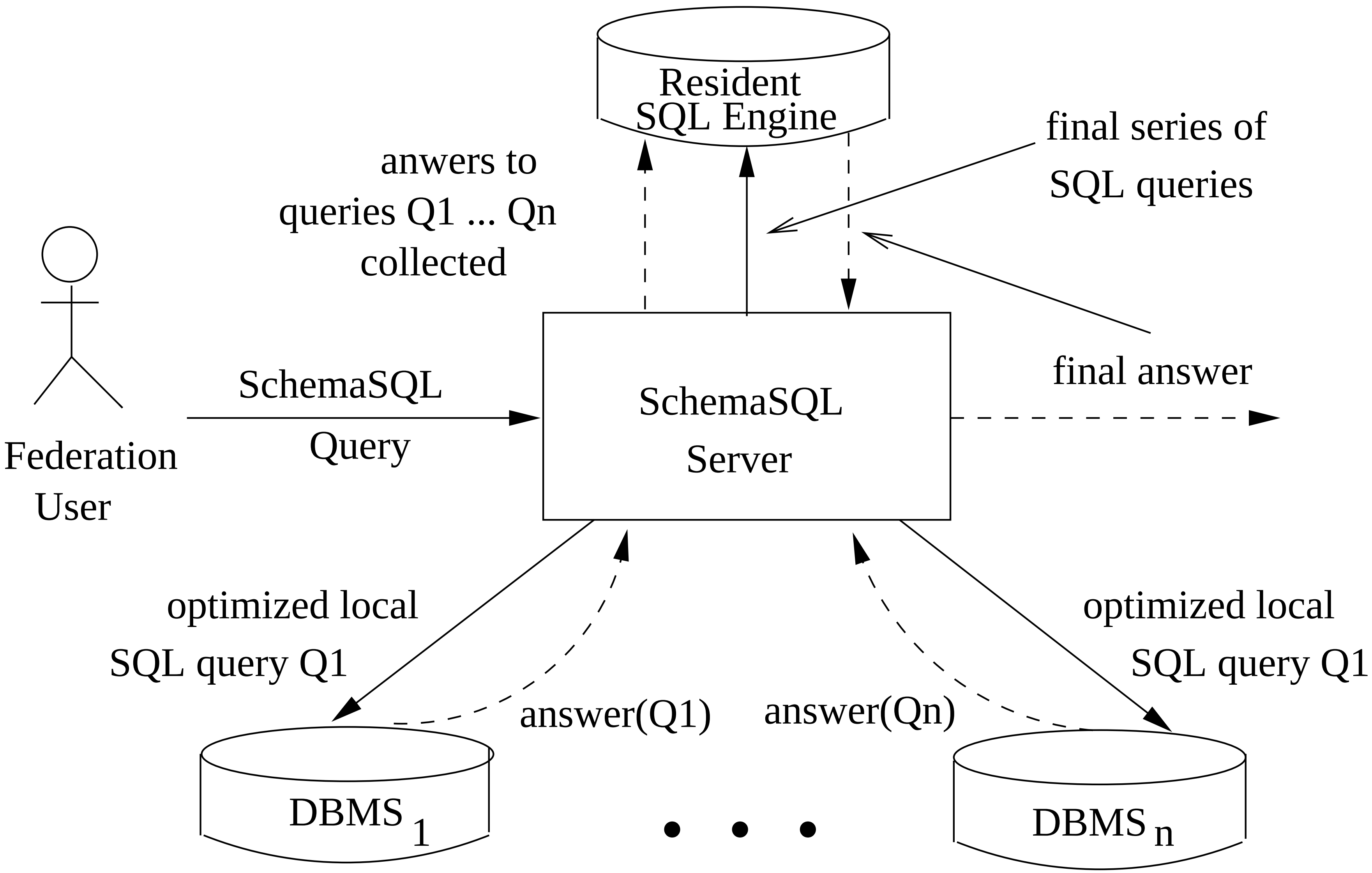
Erstellen Sie das folgende View, das die Informationen der vier Datenbanken uniform darstellt:

create view
globalSchema::salInfo(univ, dept, category, salFloor) as
[TO BE COMPLETED]

Lösung

```
create view
globalSchema::salInfo(univ, dept, category, salFloor) as
    select “univ-A”, T.dept, T.category, T.salFloor
    from univ-A::salInfo T
union
    select “univ-B”, D, T.category, T.D
    from univ-B::salInfo T, univ-B::salInfo→D
    where D<>“category”
union
    select “univ-C”, T, T.category, T.salFloor
    from univ-C→D, univ-C::D T
union
    select “univ-D”, T.dept, C, T.D
    from univ-D::salInfo T, univ-D::salInfo→C
    where C<>“dept”
```

Architektur einer Implementierung



Anfrageberechnung

Federation System Table (FST): Meta-Information über die Komponentendatenbanken, wie Namen der Datenbanken, Relationen, Attribute, oder für Anfrageoptimierung nützliche statistische Information (analog Data Dictionary).

Variable Instantiation Tables (VIT): enthalten die während der Ausführung einer Anfrage möglichen Variablenbindungen (auf Metaebene).

Input: eine *SchemaSQL*-Anfrage

Output: Bindungen der Variablen der SELECT-Klausel der Anfrage

Auswertung: Zwei Phasen:

1. Erzeugen der VITs entsprechend den Variablen in der FROM-Klausel. Dazu werden SQL-Anfragen an lokale Datenbasen sowie die FST gestellt.
2. Umschreiben der *SchemaSQL*-Anfrage in eine äquivalente Anfrage unter Verwendung der VITs (Dynamic SQL). Diese Anfrage wird dann durch den *resident SQL server* ausgewertet.

Auswertung: Beispiel

select RelC
from univ-C$\rightarrow$ RelC, univ-C::RelC C,
univ-D::salInfo D
where RelC = D.dept and
C.category = “Prof” and
C.salFloor $>$ D.Prof

Bindungen für Metavariablen (Anfrage an FST):

VIT_{RelC}
RelC
CS
Math

Bindungen für Tupelvariablen (Anfragen an Komponenten-DBS):

VIT_C (abh. von Rel_C)		
RelC	category	salFloor
CS	Prof	60,000
CS	Assoc Prof	55,000
Math	Prof	70,000
Math	Assoc Prof	60,000

VIT_D		
Dept	Prof	AssocProf
CS	75,000	60,000
Math	60,000	45,000

Auswertung: Beispiel

... nochmal die Anfrage:

select RelC
from univ-C$\rightarrow$ RelC, univ-C::RelC C,
    univ-D::salInfo D
where RelC = D.dept and
      C.category = “Prof” and
      C.salFloor $>$ D.Prof

Anfragebearbeitung mittels Standard-SQL auf den $VIT's$.

select VIT_RelC.RelC
from VIT_RelC, VIT_C, VIT_D
where VIT_C.RelC = VIT_RelC.RelC   % Korrelation RelC, C
    and VIT_RelC.RelC = VIT_D.dept
    and VIT_C.category = “Prof”
    and VIT_C.salFloor $>$ VIT_D.Prof

Aufgabe: Schema-SQL

Beschreiben Sie den Ablauf der Anfrage von Folie 67 mit FST und VITs.

VIT_{dbname}
dbname
univ-A
univ-B
univ-C
univ-D

$VIT_{relname}$	
dname	relname
univ-A	salInfo
univ-B	salInfo
univ-C	CS
univ-C	math
univ-D	salInfo

SELECT VIT_{dbname}.dbname, $VIT_{relname}$.relname
FROM VIT_{dbname}, $VIT_{relname}$
WHERE VIT_{dbname}.dbname = $VIT_{relname}$.relname

Aufgabe: Schema-SQL

- Formulieren Sie die Anfrage “An welchen Tagen hatten welche Aktien einen Kurs von 150 $?” für die auf Folie 21 angegebenen Schemata.
- In kommerziellen Datenbanksystemen sind die Schemainformationen im *Data Dictionary* abgelegt (vgl. die beiden folgenden Ausschnitte der relevanten Tabellendefinitionen):

```
SQL> desc sys.user_tables;
 Name                       Null?    Type
 -----------------------    -------- ----
 TABLE_NAME                 NOT NULL VARCHAR2(30)

SQL> desc sys.user_tab_columns;
 Name                       Null?    Type
 -----------------------    -------- ----
 TABLE_NAME                 NOT NULL VARCHAR2(30)
 COLUMN_NAME                NOT NULL VARCHAR2(30)
 DATA_TYPE                  VARCHAR2(30)
```

  Beschreiben Sie, wie die obigen Anfragen in einer Umgebung, wo SQL in eine prozedurale Programmiersprache eingebettet ist (z.B. embedded-SQL oder PL/SQL) ablaufen könnten (Pseudocode).

Lösung

- SELECT Datum, Name
  FROM Frankfurt::Kurs
  WHERE Wert=150;

  SELECT Datum, AttrName
  FROM Tokyo::Kurs.Datum,
  Tokyo::Kurs $\rightarrow$ AttrName
  WHERE AttrName $\neq$ 'Datum'
  AND Wert=150;

  SELECT NewYork::TabName.Datum, TabName
  FROM NewYork $\rightarrow$ TabName
  WHERE Wert=150;

  SELECT TabName, Sydney::TabName.Name
  FROM Sydney $\rightarrow$ TabName
  WHERE Wert=150;

- Auf die Informationen des Data Dictionary muß nur in Tokyo, New York und Sydney zugegriffen werden.

Lösung

Algorithmus für SQL in einer prozeduralen Umgebung (Datenbank Tokyo):

- Ausgabe von

  SELECT ColumnName
  FROM Tokyo.user_tab_columns
  WHERE ColumnName ≠ 'Datum';

  speichern (Ergebnis: die Namen der Firmen) und für jedes Ergebnis <cn> die Anfrage

  SELECT Datum, <cn>
  FROM Tokyo.Kurs
  WHERE <cn>= 150;

  auswerten und alle Ergebnisse sammeln.

Lösung

- Datenbank C: Ausgabe von

  SELECT TableName
  FROM user_tables
  WHERE
  (SELECT ColumnName
  FROM user_tab_columns UTC
  WHERE UTC.TableName=TableName) = {Datum,Wert};

  speichern (die verwendete Mengengleichheit muß natürlich in SQL umschrieben werden) und für jedes Ergebnis <tn> die Anfrage

  SELECT Datum, <tn>
  FROM <tn>
  WHERE Wert = 150;

  auswerten und alle Ergebnisse sammeln.

Problem hierbei: SQL-Statements müssen *dynamisch* generiert werden: Die Ergebnisse der ersten Anfrage müssen in das zweite Statement eingebaut werden.

Lösung

In Oracle (seit 7.1) ist dies z.B. mit dem DBMS_SQL-Package (in PL/SQL einzubinden) möglich, das es erlaubt, zur Laufzeit SQL-Statements zu erzeugen:

```
declare
 cursor col_cursor is
   select column_name, data_type
   from sys.user_tab_columns
   where table_name = upper('&&table_name')
   order by column_id;
 lv_column_name   sys.user_tab_columns.column_name%TYPE;
 lv_column_typ    sys.user_tab_columns.data_type%TYPE;
 lv_rowid         varchar2(20);
 rows_processed   number;
 loop_count       number;
 stmnt            varchar2(2000);
 doublecur        BINARY_INTEGER;
 execute_feedback INTEGER;
 type colname_typ is table of lv_column_name%TYPE
      index by binary_integer;
 type rowid_typ is table of lv_rowid%TYPE
      index by binary_integer;
 colname_table      colname_typ;
 empty_colname      colname_typ;
 rowid_table        rowid_typ;
 empty_rowid_table  rowid_typ;
```

```
begin
   DBMS_OUTPUT.ENABLE(10000);
   rows_processed := 0;
   -- Suche Attribute mit Datentyp "Number"
   open col_cursor;
   loop
      fetch col_cursor into
         lv_column_name, lv_column_typ;
      exit when col_cursor%notfound;
      IF lv_column_typ='NUMBER' THEN
         rows_processed := rows_processed+1;
         colname_table (rows_processed)
              := lv_column_name;
      END IF;
   end loop;
   close col_cursor;

   -- Initialisiere Abfrage-Statement
   stmnt := 'select rowid from '
       || '&&table_name '
       || 'where ';
```

```
-- Erstelle iterativ den Abfrage-Befehl
loop_count := 1;
WHILE loop_count <= rows_processed
loop
   stmnt := stmnt
       || colname_table(loop_count)
       || ' = &&Wert';
   if loop_count < rows_processed
   then
     stmnt := stmnt || ' or ';
   end if;
   loop_count := loop_count + 1;
end loop;
DBMS_OUTPUT.PUT_LINE
        ('Berechnete Abfrage: ' || stmnt);

-- Fuehre den neuen Befehl aus
doublecur := DBMS_SQL.OPEN_CURSOR;
DBMS_SQL.PARSE (doublecur
          ,stmnt
          ,DBMS_SQL.V7);
DBMS_SQL.DEFINE_COLUMN
     (doublecur, 1, lv_rowid, 20);
execute_feedback := DBMS_SQL.EXECUTE (doublecur);
```

```
   -- Erstelle Liste der gefundenen DS und
   -- RowIDs aus.
   loop
     if DBMS_SQL.FETCH_ROWS (doublecur) = 0
     then
       exit;
     else
       DBMS_SQL.COLUMN_VALUE (doublecur,1, lv_rowid);
       DBMS_OUTPUT.PUT_LINE('RowID: ' ||lv_rowid);
     end if;
   end loop;

   -- Aufraeumarbeiten und Ende
   DBMS_SQL.CLOSE_CURSOR (doublecur);
   colname_table := empty_colname;
end;
/
```

Lösung

```
SQL> @find-number;
Geben Sie einen Wert fuer table_name ein: Tokyo
Geben Sie einen Wert fuer wert ein: 150
Berechnete Abfrage:
  select rowid from Tokyo
  where SUN = 150 or DC = 150 or FUJI = 150

RowID: AAAA2MAADAAAD7nAAA

SQL> select * from Tokyo
     where rowid='AAAA2MAADAAAD7nAAA';
03.03.93        150        151        140
```

was man natürlich noch weiter nacharbeiten muss um die Antwort `'sun', 3.3.93` zu bekommen.

- Fazit: Schema-SQL macht solche Anfragen erheblich kürzer und übersichtlicher und ist einfacher zu lernen als PL/SQL und DBMS_SQL.

Aufgabe: Horizontale und blockweise Gruppierungen

- Geben Sie für die Schemata `univ-B`, `univ-C` und `univ-D` SchemaSQL-Anfragen an, die für jede Gehaltsklasse das durchschnittliche Grundgehalt über alle Departements berechnen.

Lösung: Horizontale und blockweise Gruppierungen

- univ-A: Wie in Standard SQL: vertikale Aggregierung:

  select T.category, avg(T.salFloor)
  from univ-A::salInfo T – Tupelvariable
  group by T.category

- univ-B: horizontale Aggregierung
  siehe Folie 72.

- univ-C: Aggregierung über verschiedene Tabellen
  siehe Folie 73.

- univ-D: Aggregierung in verschiedenen Spalten:

  select T.category, avg(T.C)
  from univ-B::salInfo T, univ-B::salInfo $\rightarrow$ C
  where C $<>$ “dept”
  group by C

Lösung: Horizontale und blockweise Gruppierungen

- univ-A: vertikale Gruppierung

```
SELECT T.category,avg(T.salFloor)
FROM univ-A::sal-Info T
GROUP BY T.category;
```

- univ-B: horizontale Gruppierung

```
SELECT T.category,avg(T.D)
FROM univ-B::sal-Info T,
          univ-B::sal-Info → D
WHERE D ≠ 'Category'
GROUP BY T.category;
```

- univ-C: Gruppierung über verschiedene Tabellen

```
SELECT T.category,avg(T.salFloor)
FROM univ-C → D
          univ-C::D T
GROUP BY T.category;
```

- univ-D: vertikale Gruppierung verschiedener Spalten

```
SELECT C, avg(T.C)
FROM univ-D::salInfo T
          univ-D::salInfo → C
WHERE C ≠ 'dept'
GROUP BY C;
```

Aufgabe: Schlüssel-Test

Bei der Integration verschiedener Datenbanken steht man häufig vor dem Problem, Daten aus semantisch verwandten Tabellen zu vereinigen. Dabei stellt sich für die integrierte Datenbank das Problem, welche Datensätze zusammengehören, und ob die Schlüsseleigenschaften weiterhin erfüllt sind.
Betrachten Sie zwei Relationen

$R_1 : (K_1, \ldots, K_n, A_1, \ldots, A_m, B_1, \ldots, B_p)$ und
$R_2 : (K_1, \ldots, K_n, A_1, \ldots, A_m, C_1, \ldots, C_q)$,

wobei $K_1, \ldots, K_n$ Primärschlüssel und $A_1, \ldots, A_m$ weitere gemeinsame Attribute sind. R_1 und R_2 sollen zu einer gemeinsamen Relation
$R^* : (K_1, \ldots, K_n, A_1, \ldots, A_m, B_1, \ldots, B_p, C_1, \ldots, c_q)$ vereinigt werden.

Formulieren Sie, unter welchen Bedingungen eine Vereinigung problemlos möglich ist und formulieren Sie eine SQL-Anfrage, die diese Bedingung testet und alle problematischen Tupel(paare) ausgibt. Formulieren Sie dieselbe Bedingung in Schema-SQL.

Lösung: Schlüssel-Test

Eine Vereinigung ist dann problemlos möglich, wenn für alle Paare $R_1(k_1, \ldots, k_n, a_1, \ldots, a_m, b_1, \ldots, b_p)$ und $R_2(k_1, \ldots, k_n, a'_1, \ldots, a'_m, c_1, \ldots, c_q)$ deren Schlüsselwerte übereinstimmen, für alle $1 \leq i \leq m$ gilt

$$a_i = a'_i \text{ oder } a_i = NULL \text{ oder } a'_i = NULL\ .$$

Dies wird mit der folgenden SQL-Anfrage überprüft:

```
select * from R1,R2
where R1.K1=R2.K1 and ... and R1.Kn=R2.Kn
  and((not R1.A1 is NULL and not R2.A1 is NULL
          and NOT R1.A1= R2.A1)
          or       ⋮
          or (not R1.Am is NULL and not R2.Am is NULL
            and NOT R1.Am= R2.Am));
```

In Schema-SQL ist diese Anfrage einfacher:

```
select * from R1,R2,R1 → A
where R1.K1=R2.K1 and ... and R1.Kn=R2.Kn
  and (not R1.A is NULL and not R2.A is NULL
            and NOT R1.A= R2.A);
```

(unter der Annahme daß `not R.A IS NULL` zu *false* ausgewertet wird, wenn A kein Attribut von R ist).

Weitere Funktionalität von SchemaSQL: Restrukturierung

- Daten $\Leftrightarrow$ Metadaten
- Restrukturierung von Tabellen
- $\mathcal{V}$: Werte
- $\mathcal{N}$: Namen (Tabellen und Attribute)
- für jedes $n \in \mathcal{N} : dom(\mathcal{N}) \subseteq \mathcal{V}$

Ein Datenbankschema ist *flach* wenn alle Signaturen $R(A_1, \ldots, A_n)$ nur aus Namen bestehen.

Restrukturierung ist nur für nicht-flache Schemata interessant ...

Restrukturierung

- UNFOLD <rel> BY <attr$_i$> ON <attr$_j$>
  - Die Werte von <attr$_i$> werden zu Spaltennamen,
  - <attr$_j$> fällt weg, die Werte von Attr$_j$ werden zu Einträgen der neuen Spalten

  Tokyo = UNFOLD *Frankfurt* BY *Name* ON *Wert*
  - Vorsicht: Problem, wenn es z.B. noch ein Attribut "Umsatz" gibt, das das tägliche Unmsatzvolumen jeder Aktie angibt
  - Aufgabe: Lösen Sie dieses Problem.
- FOLD <rel> BY <domain$_1$> ON <domain$_2$>
  - <domain$_2$> muß eine Menge von Namen sein, die als Spaltennamen in <rel> auftreten; <domain$_2$> wird eine neue Spalte des Ergebnisses, dafür fallen diese Spalten weg.
  - alle Werte dieser Spalten müssen aus <domain$_1$> sein, diese Werte werden in einer neuen Spalte <domain$_1$> des Ergebnisses gespeichert.

  Frankfurt = FOLD *Tokyo* BY *Wert* ON *Name*

Restrukturierung

- SPLIT <rel> BY <attr>
  - Die Spalte <attr> fällt weg, für jeden ihrer Werte wird eine neue Tabelle erstellt.

  New York = SPLIT *Frankfurt* BY *Name*
  Sydney = SPLIT *Frankfurt* BY *Datum*

- UNITE BY domain | condition
  - Alle Tabellen <rel>, so daß <rel> ∈ <domain> ist und <condition> erfüllt ist, werden vereinigt, wobei eine zusätzliche Spalte <domain> hinzugenommen wird . Sinnvoll ist es, wenn diese alle ein gemeinsames Schema besitzen.

  Frankfurt = UNITE BY *Name* | (einschr. auf New York)
  Frankfurt = UNITE BY *Datum* | (einschr. auf Sydney)

Aufgabe

Definieren Sie die oben genannten Restrukturierungsoperatoren durch Schema-SQL-Ausdrücke.

```
create view db::unfold
  (<alle attribute ausser attri und attrj>, I) as
select <alle attribute ausser attri und attrj>, attrj
from db::rel T, db::rel.attri I
      % I laeuft ueber Attributwerte von attri
where T.attri = I

create view db::fold
 (<alle attr so dass attr not in domain2>, D, T.D) as
select (<alle attr ausser attri und attrj>, D, T.D)
from db::rel T, db::rel->D
where D in domain2

create view db::A as
select <alles ausser attr>
from db::rel T, db::rel.attr A
      % A laeuft ueber Attributwerte von attr
where T.attr = A

create view db::unite as
select D, T.attr1 ... T.attrn
from db -> D, db::D T          % D: Relation
where condition(D)
```

Heterogene Datenquellen

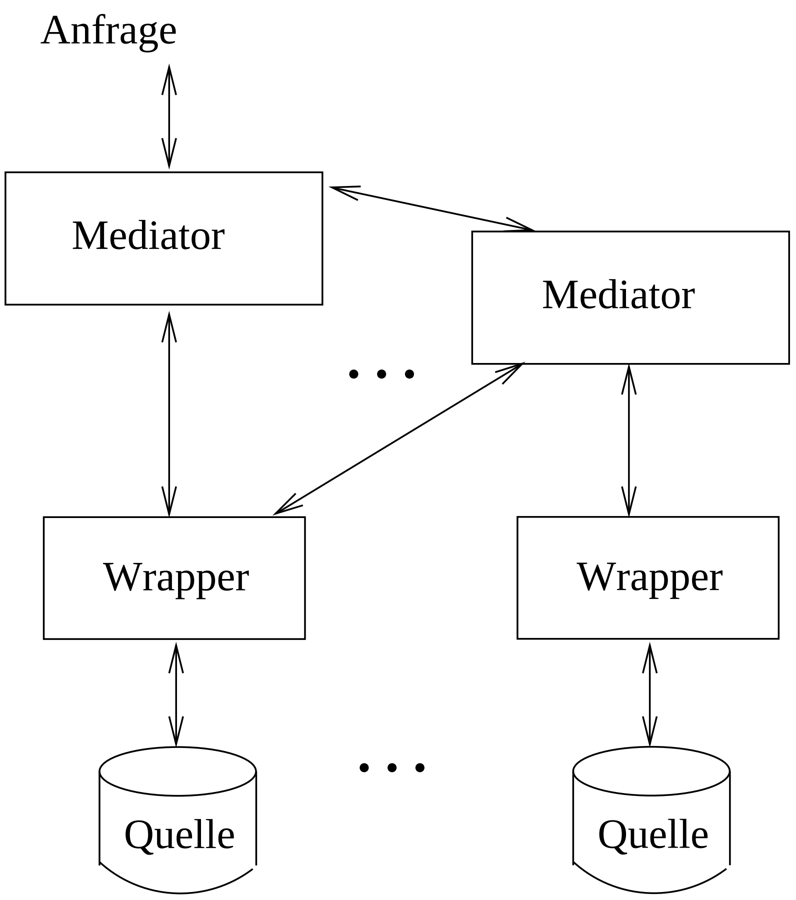
... Problematik tritt bereits bei “klassischen” Anwendungen auf (Legacy-Systeme; Netzwerk-Datenmodell etc.)

zusätzlich aktuelle Anforderungen, die über klassische föderierte Datenbanken hinausgehen (Information *Retrieval*)

- höhere (bis absolute) Autonomität
- evtl. Datenquellen nur transient verfügbar
- dafür weltweite Auswahl an Datenquellen
- verschiedene Datenmodelle, u.a. *semistructured data* (u.a. Web-Seiten)
- weltweit aktuelles Forschungsgebiet

Wrapper/Mediator-basierte Architekturen

- *Mediator* (Vermittler): Ebene zwischen Benutzern und Datenquellen (middleware),
- *Wrapper (Translator)*: bieten dem Mediator homogenen Zugriff zu heterogenen Quellen



Kommentare

- Quellen: Datenbanken, Oberflächen auf Datenbanken mit Forms/Suchmasken (z.B. Bibliotheks-Suchsysteme), Suchmaschinen, einfache Web-Seiten

- prinzipiell besitzt jede verwendete Datenquelle einen eigenen Wrapper
- Wrapper muß nicht demselben Besitzer wie die Datenbank gehören
- verschiedene Benutzer (programmieren und) verwenden verschiedene Wrapper

- Bsp. für Mediatoren: Meta-Suchmaschinen, Suchsysteme für mehrere Bibliotheken
- Mediator enthält Wissen über die zur Verfügung stehenden Datenquellen
- häufig Verwendung wissensbasierter Technologie (KI) z.B. Ontologien: Definition der verwendeten Termini um Informationen verknüpfen zu können
- Mediatoren können hierarchisch angeordnet sein.

Wrapper/Mediator-basierte Architekturen

Warehouse: Bottom-up

- Wrapper bilden den Inhalt der Datenquellen in ein allgemeines Datenmodell ab,
- Mediator integriert dies zu einer globalen Datenbasis
- Benutzer stellt Anfragen an den Mediator, die auf der globalen Datenbasis ausgewertet werden.

Virtual: Top-Down

- Mediator stellt eine allgemeine Anfragesprache zur Verfügung
- Benutzer stellt Anfragen an den Mediator, die von diesem aufgeteilt werden und die Teile an die Wrapper weitergegeben werden.
- Wrapper stellen die Anfragen an die Teildatenbasen, und geben Ergebnisse zurück
- Mediator verknüpft Ergebnisse.

Wrapper/Mediator-basierte Architekturen

Warehouse: Bottom-up

– erfordert extensionale globale Datenbank

– erschwerte Reaktion auf Veränderungen der Originaldatenbanken

+ toleranter gegenüber Ausfällen von Teildatenbanken

+ i.a. kürzere Antwortzeit

Virtual: Top-Down

+ Antwort entspricht immer dem aktuellen Datenbestand

– evtl. lange Berechnungszeiten

– empfindlich gegenüber Ausfällen

TSIMMIS

(The Stanford-IBM Manager of Multiple Information Sources, bzw. Jiddisch “Eintopf”)

URL: http://www-db.stanford.edu/tsimmis/tsimmis.html

Projektziel (mehrere Teilprojekte): Schaffung von Hilfsmitteln zum konsistenten integrierten Zugriff zu Informationsquellen:

- Heterogene Informationsquellen
  - Datenbanken
  - Filesystem
  - Web-Seiten

  ⇒ häufig kein explizites Schema vorhanden

  ⇒ Abbildung auf ein Common Data Model

- Inhalt der Quellen kann sich ändern

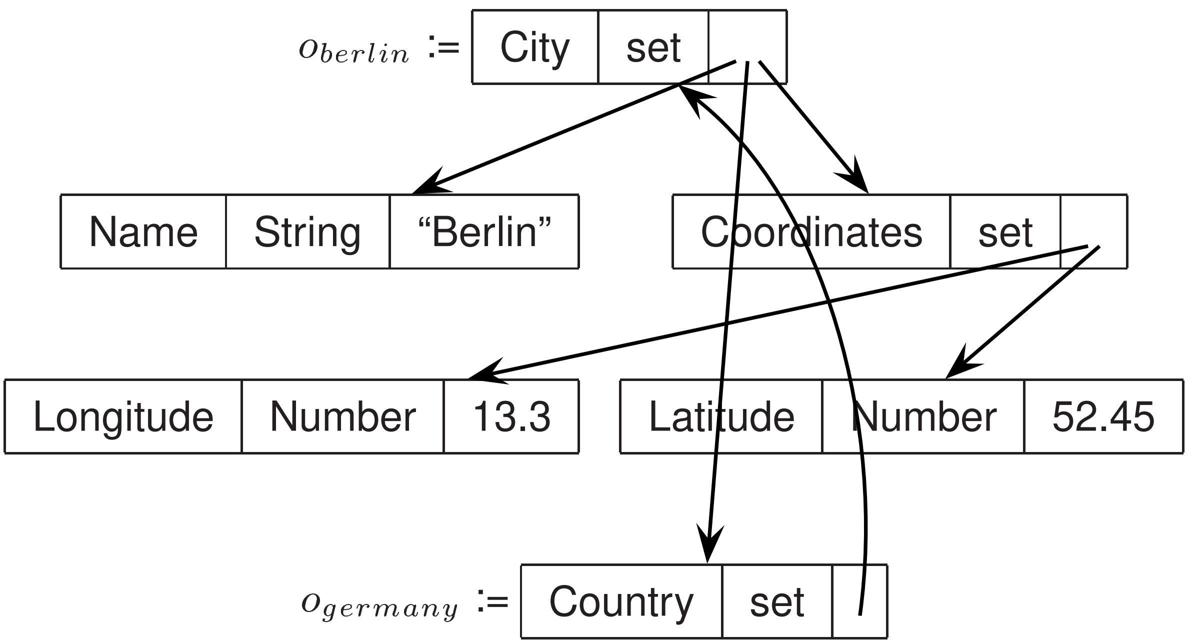
TSIMMIS: Konzept

- Virtual Approach:
- Benutzer stellt Anfrage an einen Mediator
- Mediator gibt Anfrage an die Wrapper (“Translatoren”) weiter
- Translatoren sind Programme, die die Datenobjekte der Quellen (logisch) in OEM transformieren und die Anfragen aus den Quellen beantworten können
- Resultate der Quellen werden in OEM an den Mediator zurückgegeben
- Mediator integriert die Antworten der Quellen.
- Mediatoren können hierarchisch angeordnet sein.

OEM: Object Exchange Model

Datenmodell zur Kommunikation zwischen Wrappern und Mediatoren in TSIMMIS

- sehr einfaches, “selbstbeschreibendes” Objektmodell
- kennt nur Objektidentität und Schachtelung als Konzepte:
- Jedes Objekt hat eine Objekt-ID, ein Label (entsprechend etwa einer Klasse), einen Typ und einen Wert,
- Werte komplexer Typen sind Mengen von Referenzen auf Subobjekte
- Labels: “selbstbeschreibende Daten”
- top-level Objekte mit semantischen Objektbezeichnern als Einstiegspunkte (vgl. OQL)
- als Graph darstellbar:



TSIMMIS/OEM

Quelle 1: CIA World Factbook

Wrapper cia exportiert OEM-Objekte wie folgt:

<&cont1, continent, set, {&a1, &n1, &c1, &c2, &c3, . . . }>
<&n1, name, string, `Europe'>
<&a1, area, number, 9562488>
<&c1, country, set, {cn1, cc1, ca1, cp1, cap1}>
<&c2, country, set, {cn2, cc2, ca2, cp2, cap2}>

<&cn1, name, string, `Germany'>
<&cc1, code, string, `D'>
<&ca1, area, number, 356910>
<&cp1, population, number, 83536115>
<&cap1, name, string, `Berlin'>

<&cn2, name, string, `Sweden'>
<&cc2, code, string, `S'>
<&ca2, area, number, 449964>
<&cp2, population, number, 8900954>
<&cap2, name, string, `Stockholm'>

⋮

TSIMMIS/OEM

Quelle 2: Global Statistics

Wrapper gs exportiert OEM-Objekte wie folgt:

<&cont1, continent, set, {&a1, &n1, &c1, &c2, &c3, . . . }>
<&n1, name, string, `Europe'>
<&c1, country, set,
{cn1, ct11, ct12, ct13, . . . , prov11, prov21,. . . }>
<&c2, country, set,
{cn2, ct21, ct22, ct23, . . . , prov12, prov22,. . . }>

<&cn1, name, string, `Germany'>
<&ct11, city, set, {ctn11, ctp11, prov11}>
<&prov11, province, set, {pn11, pa11, pp11, ct11}>

<&ctn11, name, string, `Stuttgart'>
<&ctp11, population, number, 588482>

<&pn11, name, string, `Baden-Württemberg'>
<&pa11, area, number, 35742>
<&pp11, population, number, 10272069>

TSIMMIS: Sprachen

Mediatoren werden in MSL (Mediator Specification Language; eine regelbasierte Anfragesprache an OEM) programmiert:

MSL-Regeln:

head(Vars) :- body(Vars,databases)

- head und body bestehen aus Ausdrücken über Patterns der Form

  <oid label type value>
  oder
  <oid label value>
  oder
  <label value>

- *value* kann mengenwertig sein; in diesem Fall besteht die Menge wieder aus Ausdrücken der Form $<\ldots>$.
- Objekte der verschiedenen Quellen werden durch <objekt>@source identifiziert.
- $body$: Pattern, das mit geeigneten Variablenbindungen erfüllt werden muß,
- $head$ beschreibt die Struktur des OEM-Objekts, das generiert werden soll.

MSL

Das Land, das “D” als Code besitzt:

C :- <C country {<code “D”>}>@cia

- Anfrage erzeugt ein mengenwertiges answer-Objekt, dessen Subobjekte die einzelnen Antworten sind.

Ergebnis: <answer {&c1}>

Die Namen aller südamerikanischen Länder, in denen es eine Stadt namens Santiago gibt:

- Annahme: *southamerica* als top-level-Objekt zugreifbar

```
<countryname N>:-
<southamerica {<country
        {<name N> <city{<name “Santiago”>}>}>}>@gs
```

```
<&obj42, answer, set, {<countryname “Chile”>,
                       <countryname “Paraguay”>,
                       <countryname “Argentinien”>}>
```

Beispiel

Mediator med hat Zugriff zu Wrappern cia und gs.

Anfrage: alle Städte die in cia als capital-Name auftreten und in gs gespeichert sind:

Mediator-Regel:

<capital {<name Cap> <country CN> R }>@med :-
<country {<name CN> <capital Cap>}>@cia
AND <country {<name CN> <city {<name Cap> |
R}>}>@gs

- R wird an die restlichen Subobjekte des gefundenen city-Objektes gebunden.

⇒ Objekterzeugung im Regelkopf $obj@med$ (vgl. Views)

exportiertes Objekt z.B.

<&cap, capital, set, {&ctn12, &c1, &ctp12, &ctprov12}>@med

- außerdem: externe Prädikate zugelassen (Stringverkettung, Substring, num. Vergleiche etc).
- Variable können an Labels und an Werte gebunden werden (vgl. SchemaSQL)
- *syntaktisch* 2.Ordnung.

⇒ Anfragen an das logische Schema möglich

TSIMMIS: Sprachen

Wrapper werden in WSL (Wrapper Specification Language) programmiert,

Jede WSL-Regel besteht aus einem

- “WSL-template” (quasi eine parametrisierte MSL-Anfrage), und
- einer Aktion in der Anfragesprache der zugrundeliegenden Datenbank

Bekommt der Wrapper eine MSL-Anfrage, die auf das WSL-Template paßt, wird die Aktion mit der entsprechenden Variablenbindung ausgeführt.

Wrapper für eine SQL-Datenbank:

```
C :- <C country {<code $CODE>}>
    //$$ := “select name
              from country
              where code=$CODE”//
```

Wrapper für ein Such-Formular:

```
C :- <C country {<code $CODE>}>
    //$$ := “find-code $CODE”//
```

... Quelle kann auch ein ASCII-File sein (⇒ perl-Skript)

TSIMMIS: Sprachen

Benutzer können Anfragen in MSL oder LOREL stellen.

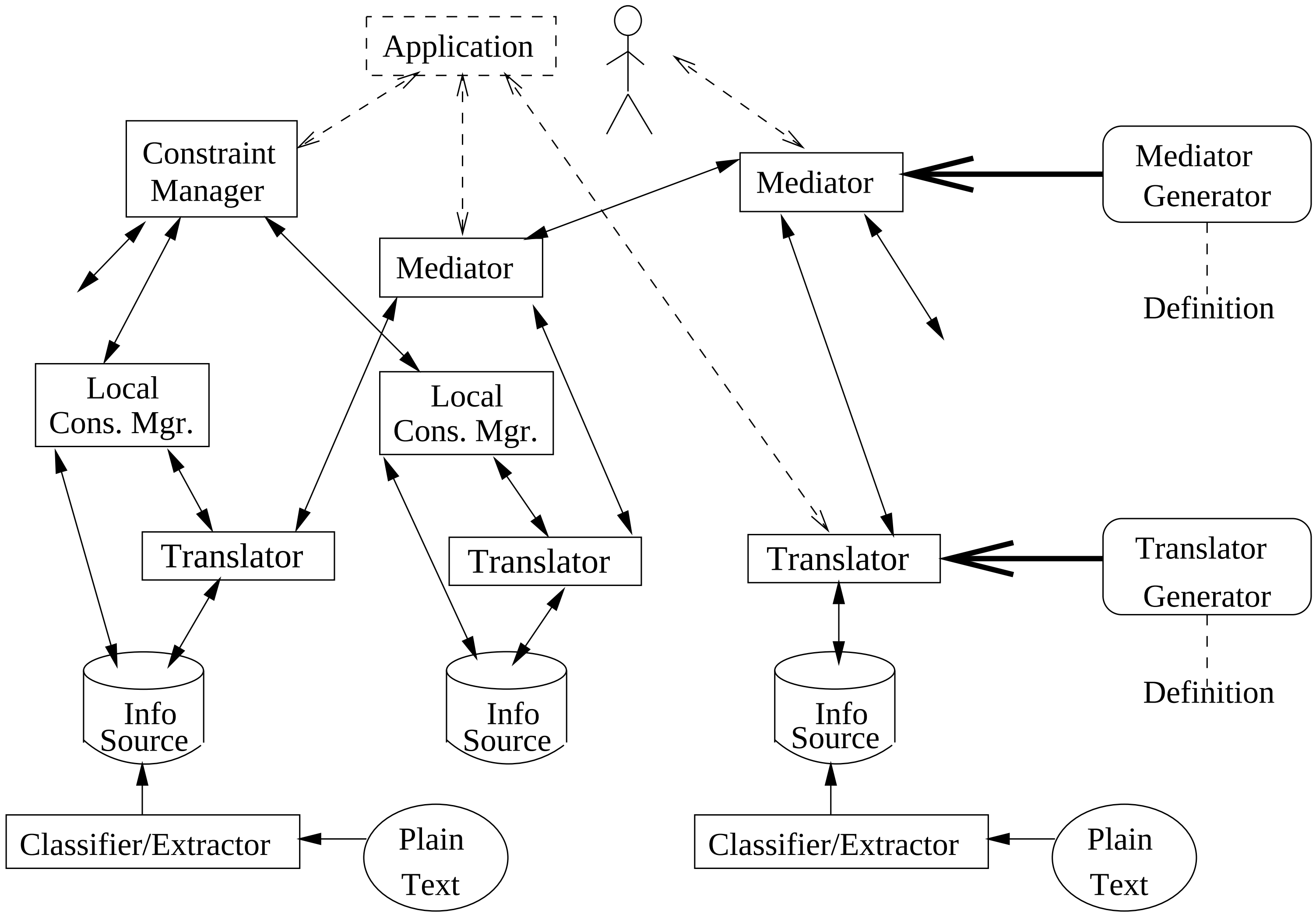
- MSL: siehe oben – regelbasierte Sprache
- LOREL (Lightweight Object Repository Language):
  (LORE: DBMS für OEM-Datenmodell, Stanford)
  OQL-basierte Anfragesprache
  Navigation im OEM-Modell durch Pfadausdrücke.

```
% alle europ. Hauptstädte:
select europe.country.capital

% Das Land mit dem Code “S”:
select europe.country
where europe.country.code = “S”

% Südamerikanische Länder mit ...
select country
from southamerica    % Top-Level-Objekt benennen
where country.city.name = “Santiago”
```

Architektur von TSIMMIS



TSIMMIS: Beispiel/Aufgabe

Stellen Sie die Universitäten-Datenbasen von Folie 64 in OEM dar.

Schreiben Sie Mediator-Regeln, die eine integrierte Sicht auf die Daten erzeugen.

COIN (Context Interchange)

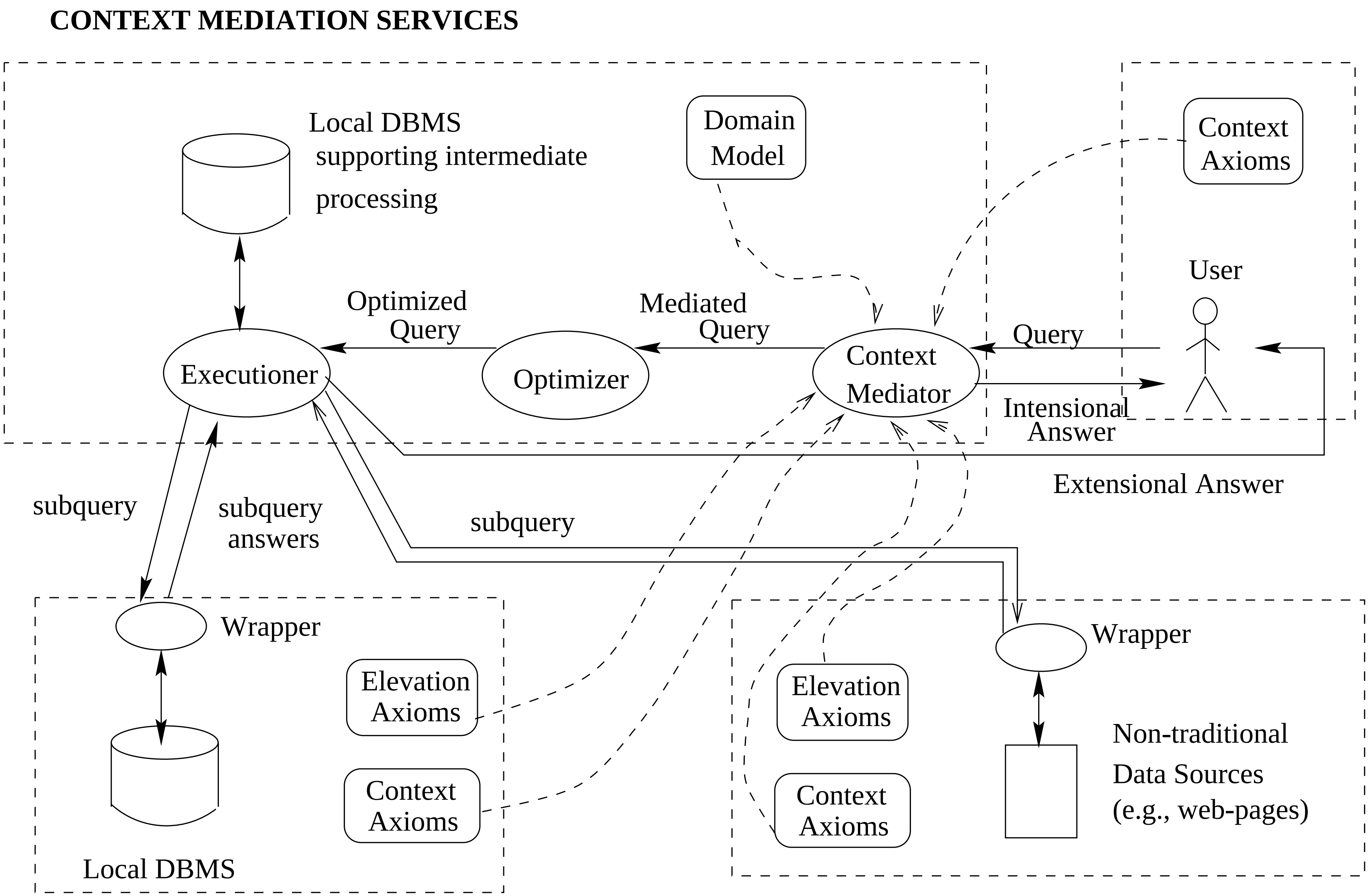
URL: http://context.mit.edu/ coin/

- **Semantische Konflikte** zwischen Quellen müssen nicht a priori bekannt sein.
- Sie werden beim Zugriff entdeckt und aufgelöst durch einen **Kontext-Mediator**, der die den Quellen und den Benutzern zugeordnete Kontextinformation analysiert.
- **Kontextinformationen** definieren, wie die Daten einer Quelle interpretiert werden sollen und wie Konflikte, wenn entdeckt, aufgelöst werden sollen.
- Einordnung: die Architektur von COIN liegt zwischen dem eng gekoppelten und dem lose gekoppelten Föderationsansatz.

Komponenten:

- **Domain Model:** globale semantische Typen und ihre Signaturen zur Definition der Anwendungswelt;
- **Elevation Axioms:** gleichen die Typen der Quellen dem Domain Modell an; bilden Objekte der Quellen auf Objekte des Domain-Modells ab.
- **Context Axioms:** definieren alternative Interpretationen der globalen Typen bzgl. der Kontexte der Quellen.

Architektur von COIN



COIN: Beispiel

Kontext c1: (Alle Beträge in Landeswährung;
Skalierungsfaktor für JPY: 1 000).

r1		
cname	revenue	country
IBM	1 000 000	USA
NTT	1 000 000	JPN

r4	
country	currency
USA	USD
JPN	JPY

Kontext c2: (Alle Beträge in USD).

r2	
cname	expenses
IBM	1 500 000
NTT	5 000 000

r3		
fromCur	toCur	exRate
USD	JPY	104.0
JPY	USD	.0096

Anfrage

SELECT r1.cname, r1.revenue FROM r1, r2
WHERE r1.cname $=$ r2.cname
AND r1.revenue $>$ r2.expenses;

Anfrage ausgeführt durch Mediator

- berücksichtigt unterschiedliche Kontexte (Umrechnungstabelle r4)

```
SELECT r1.cname, r1.revenue FROM r1, r2, r4
WHERE r1.country = r4.country AND r4.currency = 'USD'
    AND r1.cname = r2.cname AND r1.revenue > r2.expenses;
UNION
SELECT r1.cname, r1.revenue * 1000 * r3.exRate
FROM r1, r2, r3, r4
WHERE r1.country = r4.country AND r4.currency = 'JPY'
    AND r1.cname = r2.cname AND r3.fromCur = 'JPY'
    AND r3.toCur = 'USD'
    AND r1.revenue * 1000 * r3.exRate > r2.expenses
UNION
SELECT r1.cname, r1.revenue * r3.exRate
FROM r1, r2, r3, r4
WHERE r1.country = r4.country AND r4.currency <> 'USD'
    AND r4.currency <> 'JPY' AND r3.fromCur = r4.currency
    AND r3.toCur = 'USD' AND r1.cname = r2.cname
    AND r1.revenue * r3.exRate > r2.expenses;
```

Grundlagen von COIN

kontextabhängiger Wahrheitswert: Aussagen über die (Real-) Welt sind nie immer wahr oder falsch: der Wahrheitswert ist immer bzgl. eines Kontextes zu sehen.

Sei c ein Kontext, τ eine Aussage und ist (is true) ein Prädikat:

$$\overline{c} : ist(\overline{c}, \tau)$$

bedeutet τ ist wahr im Kontext c, wobei $\overline{c}$ ein **äußerer** Kontext ist.

$\overline{c} : ist$(c1, r1(“NTT“, 1.000.000, “JPY“))

$\overline{c} : ist$(c2, r2(“NTT“, 5.000.000))

Die Angabe von $\overline{c}$ kann im weiteren entfallen, da immer der Integrationskontext gemeint ist.

Grundlagen von COIN

semantische Typen:

- **primitive Objekte** (von einem primitiven Typ wie string, integer, real, ...),
- **semantischen Objekten** (von einem semantischen Typ)

- semantische Objekte haben eine kontextweite eindeutige Identität (OID).
- Auf ihnen sind Werte und Methoden definiert, die ihre kontextabhängige Semantik definieren
- is-a-Hierarchie

Elevation-Axiome: Abbildung von Werten der einzelnen Quellen auf Objekte des Domain Models.

Kontext-Axiome und Konvertierungsfunktionen: In einem Kontext implizit vorhandene semantische Zusammenhänge werden in Form von **Kontext-Axiomen** explizit gemacht. Desweiteren werden **Konvertierungsfunktionen** zwischen den einzelnen Kontexten angegeben.

Information Manifold (IM)

url: http://www.research.att.com/ levy/imhome.html

- Fähigkeit komplexe Anfragen zu beantworten, die die Kombination von Informationen unterschiedlicher Quellen erfordern.
- Benutzer werden von der Aufgabe, die relevanten Informationsquellen zu finden, zu diesen zuzugreifen und die entsprechenden Antworten zu integrieren, befreit
- Grundlage des Systems ist ein einheitliches Weltmodell.
- Das System besitzt explizite Beschreibungen des Informationsgehaltes der Quellen; diese sind Grundlage der Anfragebearbeitung (vgl. Mediator).
- Jede Quelle enthält eine Beschreibung, welche Anfragen beantwortet werden können.
- Diese Beschreibungen sind als Sichten über dem Weltmodell formuliert; Hinzunahme und Streichung von Quellen lassen das Weltmodell unberührt.

Beispiel

Informationsquellen

Source 1: Used cars for sale.
Accepts as input a category or model of car, and optionally a price range and a year range.
For each car that satisfies the conditions, gives model, year, price, and seller contact information.

Source 2: Luxury cars for sale. All cars in this database are priced above $20,000
Accepts as input a category of car and an optional price range.
For each car that satisfies the conditions, gives model, year, price, and seller contact information.

Source 3: Vintage cars for sale (cars manufactured before 1950).
Accepts as input a model and an optional price range.
Gives model, year, price, and seller contact information for qualifying cars.

Source 4: Motorcycles for sale.
Accepts as input a model and an optional price range.
Gives model, year, price, and seller contact information.

Source 5: Car reviews database. Contains reviews for cars manufactured after 1990.
Accepts as input a model and a year.

Weltmodell

- beschreibt die Anwendungsdomäne der Informationsquellen

Datenmodell

Klassen: werden durch unäre Prädikate repräsentiert; z.B. $Car(c)$.

Attribute: werden durch binäre Prädikate repräsentiert, wobei das erste Argument der Klasse entspricht; z.B. $Model(c, m)$.

ISA–Hierarchie: Auf Klassen ist eine partielle Ordnung $<$ definiert. Gilt $C < D$, dann gilt eine entsprechende Inklusionsabhängigkeit, z.B. $Car \subseteq Automobile$.

Relationen: können eine beliebige Stelligkeit haben; z.B. $ProductReview(m, y, r)$.

Weltmodell

Class	Subcl. of	Attributes	Disjoint from
Product		Model	Person
Vehicle	Product	Model, Year, Category	Stereo
Motorcycle	Vehicle	Model, Year, Category	Car
Car	Vehicle	Model, Year, Category	Motorcycle
NewCar	Car	Model, Year, Category	UsedCar
UsedCar	Car	Model, Year, Category	NewCar
CarForSale	Car	Model, Year, Category Price, SellerContact	

Relation: $ProductReview(Model, Year, Review)$

Anfrage:

Q: $q(m, p, r) \quad \leftarrow \quad CarForSale(c), Category(c, sportscar),$
$Year(c, y), y \geq 1992, Price(c, p),$
$Model(c, m), ProductReview(m, y, r)$

Beschreibung der Informationsquellen

Inhalt: wird durch eine **konjunktive Anfrage** über dem Weltmodell definiert in dem Sinn, daß alle Tupel der Quelle diese Bedingungen erfüllen;

Beispiel: eine Datenquelle, die Gebrauchtwagen enthält:

$V_1(c) \subseteq CarForSale(c), usedCar(c)$.

Anfragepotential: wird durch einen Capability Record der Form

$$(S_{in}, S_{out}, S_{sel})$$

definiert:

- S_{in}: Menge der möglichen Eingabeparameter
- S_{out} Menge der möglichen Ausgabeparameter
- $S_{sel} \subseteq S_{in} \cup S_{out}$ Parameter, auf denen Selektionsprädikate (z.B. $<$) definiert sein können.

Beschreibung der Quellen

Source 1: Used cars for sale.
Contents: $V_1(c) \subseteq CarForSale(c), UsedCar(c)$
Capabilities:
$(\{Model(c), Category(c), Year(c), Price(c)\},$
$\{Model(c), Year(c), Price(c), SellerContact(c)\},$
$\{Year(c), Price(c)\})$

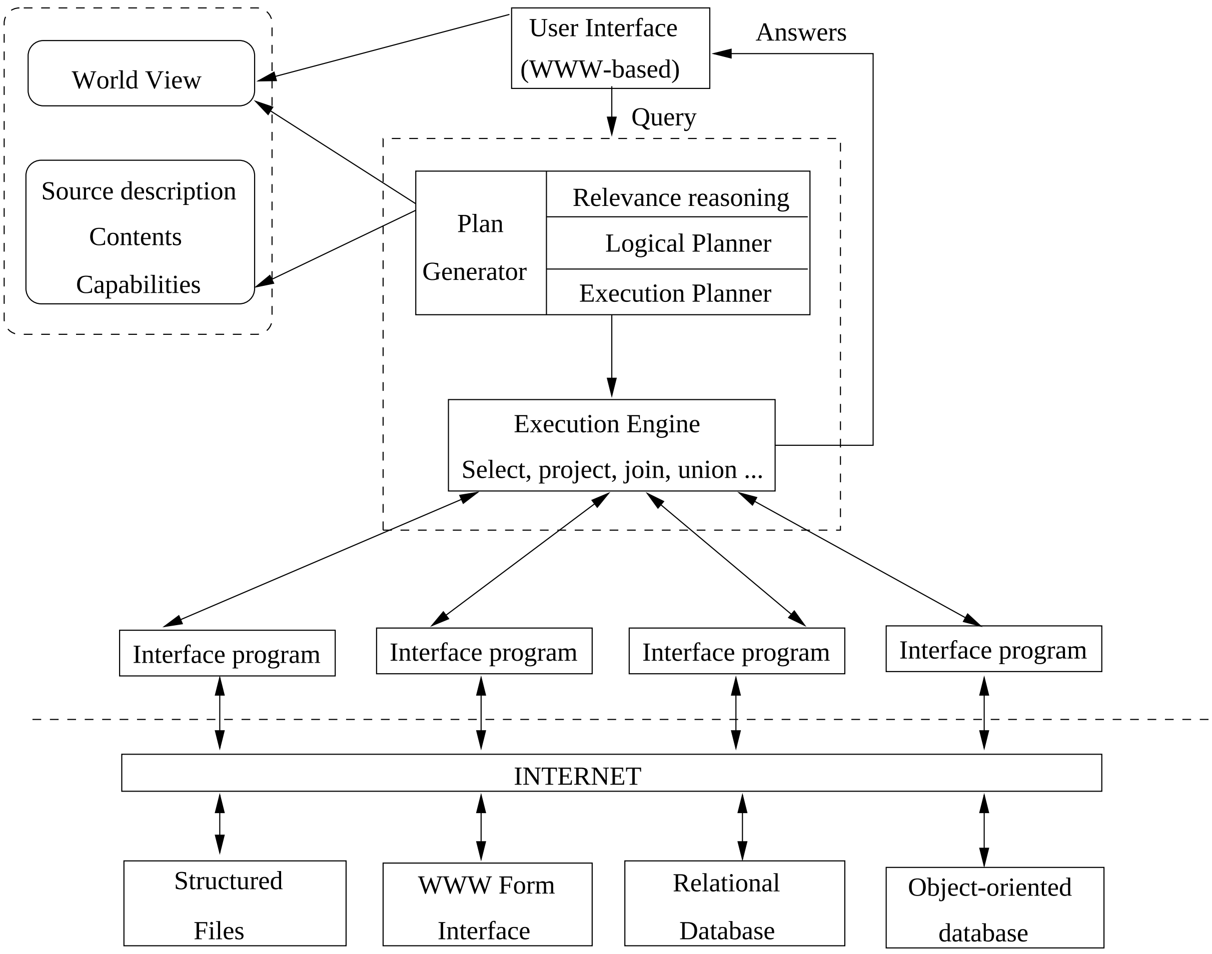
Source 2: Luxury cars for sale; priced above \$20,000
Contents: $V_2(c) \subseteq CarForSale(c), Price(c, p), p \geq 20000$
Capabilities: $(\{Category(c), Price(c)\}$
$\{Model(c), Year(c), Price(c), SellerContact(c)\}, \{Price(c)\})$

Source 3: Vintage cars for sale (manufactured before 1950).
Contents: $V_3(c) \subseteq CarForSale(c), Year(c, y), y \leq$ 1950
Capabilities: $(\{Model(c), Price(c)\},$
$\{Model(c), Year(c), Price(c), SellerContact(c)\}, \{\})$

Source 4: Motorcycles for sale.
Contents: $V_4(c) \subseteq Motorcycle(c)$
Capabilities: $(\{Model(c), Price(c)\},$
$\{Model(c), Year(c), Price(c), SellerContact(c)\}, \{Price(c)\})$

Source 5: Car reviews database. Contains reviews for cars manufactured after 1990.
Contents: $V_5(m, y, r) \subseteq$
$Car(c), Model(c, m), Year(c, y), ProductReview(m, y, r)$
Capabilities: $(\{Model(c), Year(c)\}, \{Review(c)\}, \{\})$

Architektur von IM



InfoSleuth (Info-Detektiv)

url:
http://www.mcc.com:80/projects/infosleuth/introduction/index.html

Ziel: Erforschung neuer Technologien und ihre Integration zu einem System, das in einem sich permanent ändernden Netzwerk von Informationsquellen Informationen beschafft und verarbeitet.

InfoSleuth implementiert kooperierende **Software-Agenten**, die im Auftrag von Benutzern oder Anwendungen Informationen suchen/entdecken, integrieren und präsentieren. Informationen sind verteilt

- in einem Intranet oder dem WWW

und schwierig auszuwerten aufgrund

- heterogener und fehlender Struktur,
- unterschiedlicher Ontologien (Begriffe),
- Dynamik: Quellen erscheinen und verschwinden bzw. ändern sich,
- impliziter Darstellung.

Info Sleuth: Agenten

User Agent: Intelligenter Zugang zum InfoSleuth für den Benutzer, der bei der Anfrageformulierung und Antwortdarstellung unterstützt. Ein User Agent ist persistent, so daß der Kontext des Benutzers erhalten bleibt.

Broker Agent: bringt Agenten, die einen Dienst suchen mit denjenigen zusammen, die diesen anbieten. Neue Agenten geben ihre Dienste dem Broker bekannt, einschließlich der verstandenen Sprache und Arten von beantwortbaren Anfragen.

Ontology Agent: stellt die für einen Anwendungsbereich relevanten Konzepte (Schemata) zur Verfügung. Die Benutzer werden so bei der Formulierung von Anfragen unterstützt.

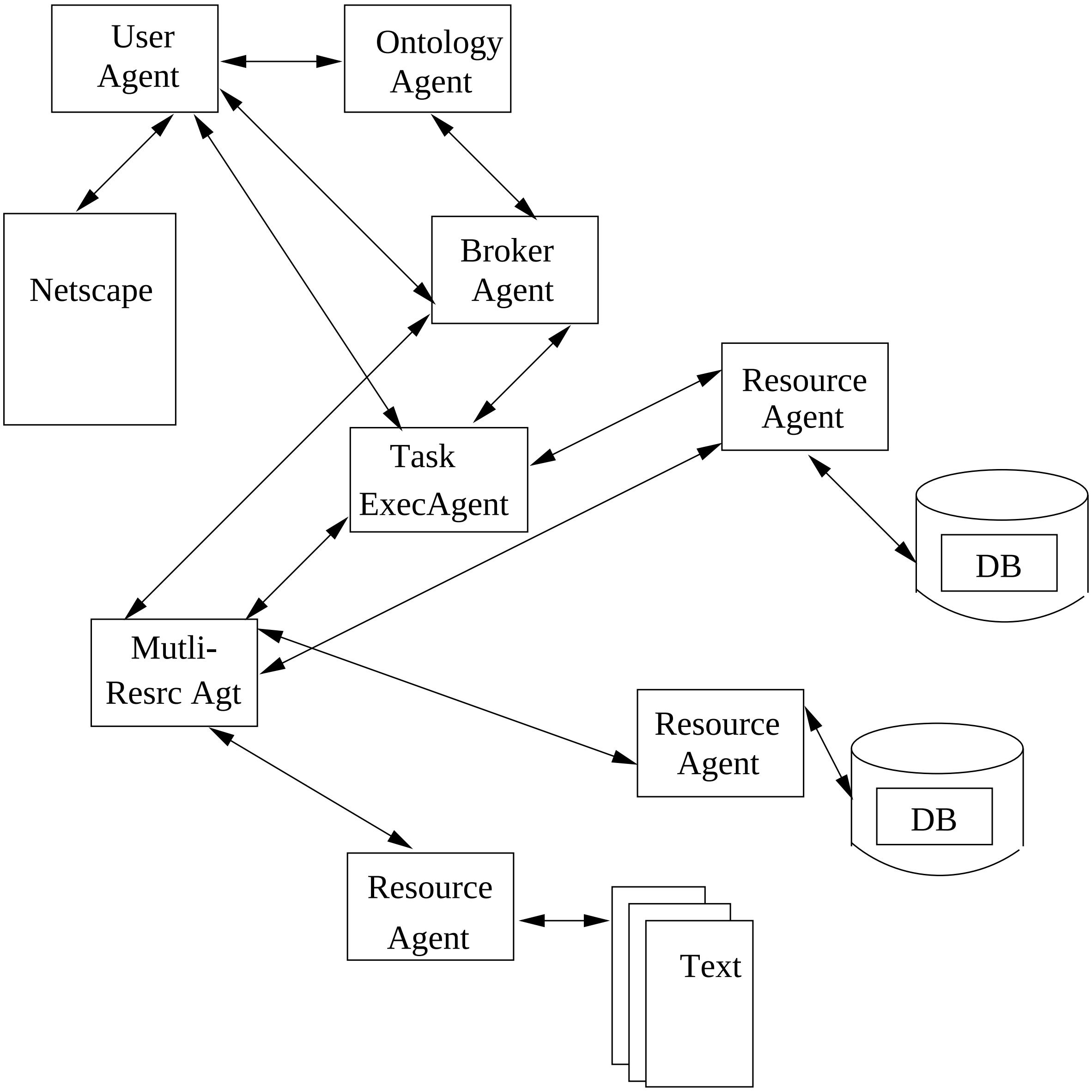
Task Execution Agent: plant und koordiniert die Bearbeitung der Dienstanforderungen der Agenten.

Resource Agent: Interface zwischen Informationsquellen und InfoSleuth.

Multiresource Query Agent: kann komplexe Anfragen in Subanfragen zerlegen und an geeignete Agenten delegieren.

Monitor Agent: protokolliert den Nachrichtenaustausch der Agenten untereinander. Grundlage von Analysen.

Architektur von InfoSleuth



FLORID

F-**LO**gic **R**easoning **I**n **D**atabases

URL: http://www.informatik.uni-freiburg.de/ dbis/florid/

- **F-Logic**: logikbasierte ("deduktive"), objekt-orientierte Datenbank-Anfragesprache
- Erweiterung zum Zugriff auf Webseiten
- Einheitliches Datenmodell sowie einheitliche logische Sprache für Wrapper, Mediator, und Benutzeranfragen
- Einheitliches (objektorientiertes) Datenmodell für Struktur *und* Inhalt des WWW

⇒ Zugriff + Verarbeitung von Information sind verflochten