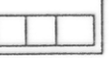
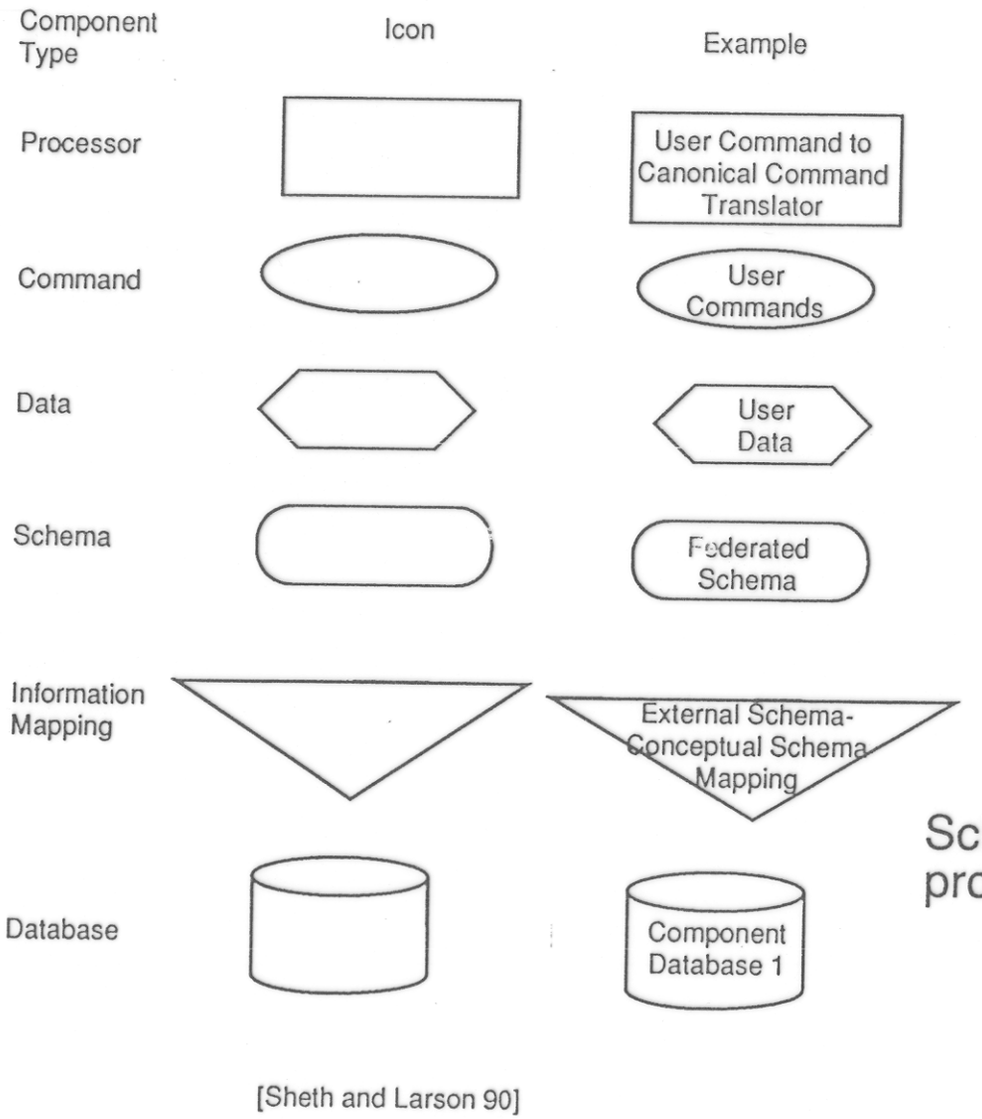
Introduction

Data in most large institutions are managed by heterogeneous and autonomous application systems and their databases. It is well understood that data is an institutional resource. However, supporting the desired level of interoperability, uniform access, and/or consistency of data will likely remain technically and organizationally a challenging task.

In this tutorial, we will study the issues and the techniques that can support interoperability in multidatabase systems from both, database and application perspectives.



ARCHITECTURE



[Sheth and Larson 90]

SYSTEM ARCHITECTURE

=

Schema Architecture

+

Processor Architecture

+

.....

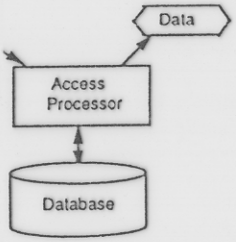
Schema are application specific, processors are application independent

Constructing Processors

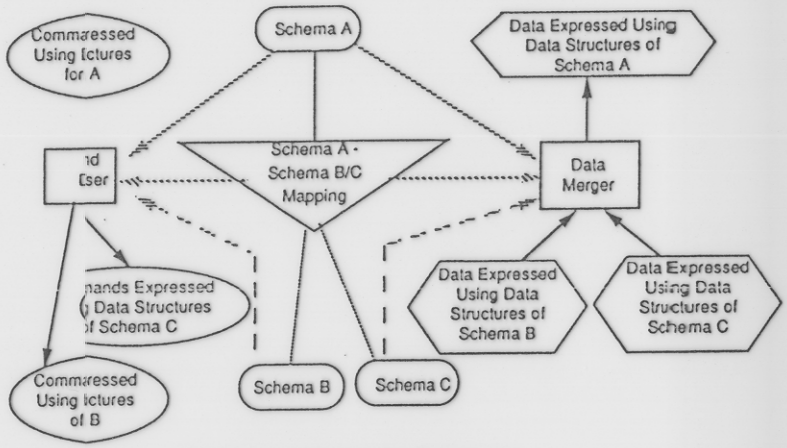
icting processors partition and replicate operations produced by a
rocessor into operations that are accepted by two or more
ors. They also merge data produced by several processors. They
port location, distribution and replication transparencies.
ery decomposer and optimizer, global transaction manager.]

Processor

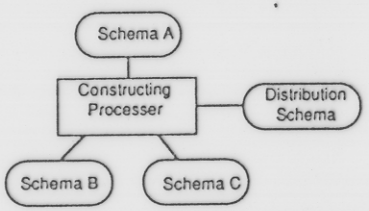
ls commands and may produce
ds against a database.
ary manager.]



and Larson 90]



(a) A pair of Constructing Processors.

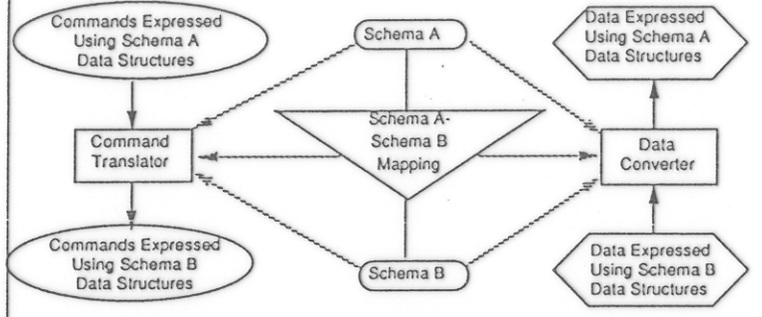


(b) An Abstract Constructing Processor.

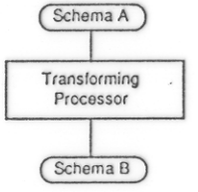
[Sheth and Larson 90]

Transforming Processors

Transforming processors translate commands (or operations) from one language or format to another language or format, or transform data from one format to another. They can be used to support data model transparency.
[E.g., A SQL- CODASYL command transformer.]



(a) A pair of companion Transforming Processors

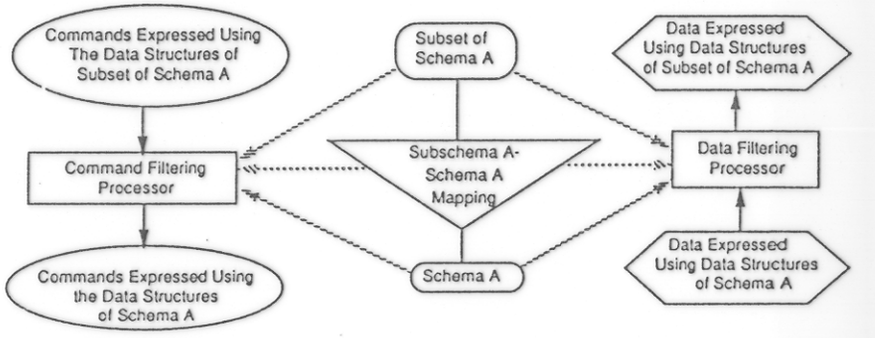


(b) An abstract Transforming Processor

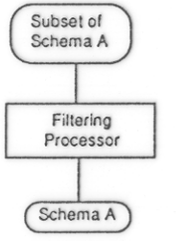
[Sheth and Larson 90]

Filtering Processors

Filtering processors constrain operations that can be applied, or constrains data that can be passed to another processor. This gives ability to control access of one processor to another.
[E.g., access controller.]



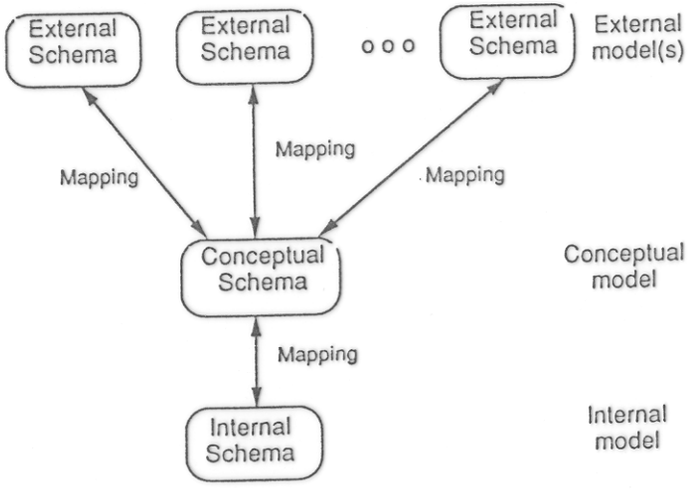
(a) A pair of companion Filtering Processors



(b) An Abstract Filtering Processor.

[Sheth and Larson 90]

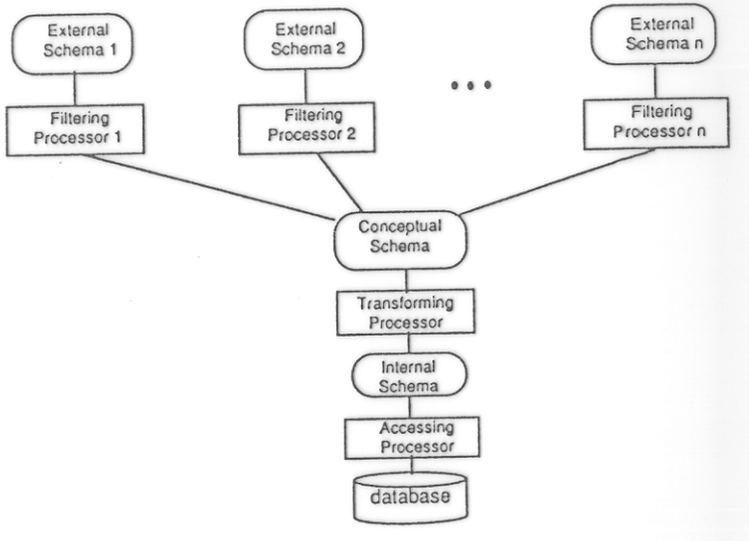
Three-Level Schema Architecture of a Centralized DBMS



The ANSI/SPARC information architecture for databases uses three schemas: internal, which describes the data storage structure and access paths; conceptual, which includes restrictions on values or relationships; and external, which describes the differesnt user views. Mapping operations relate schemas at different levels.

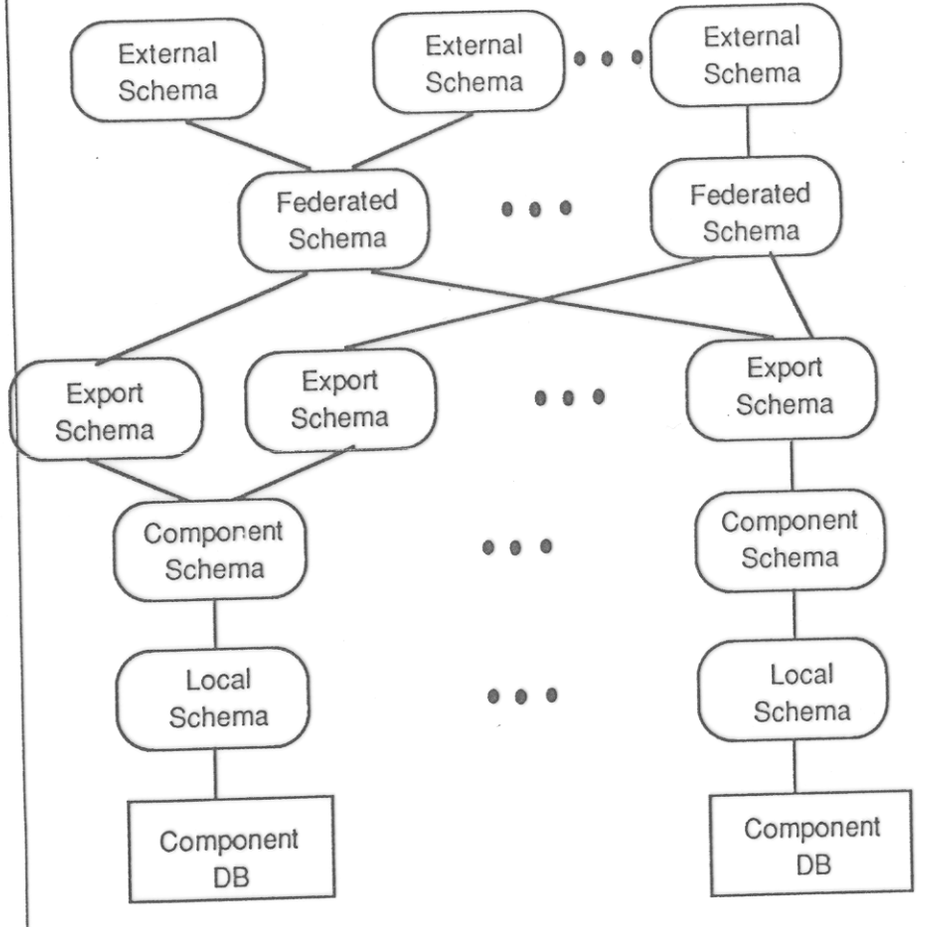
[ANSI/SPARC 77]

System Architecture of a Centralized DBMS



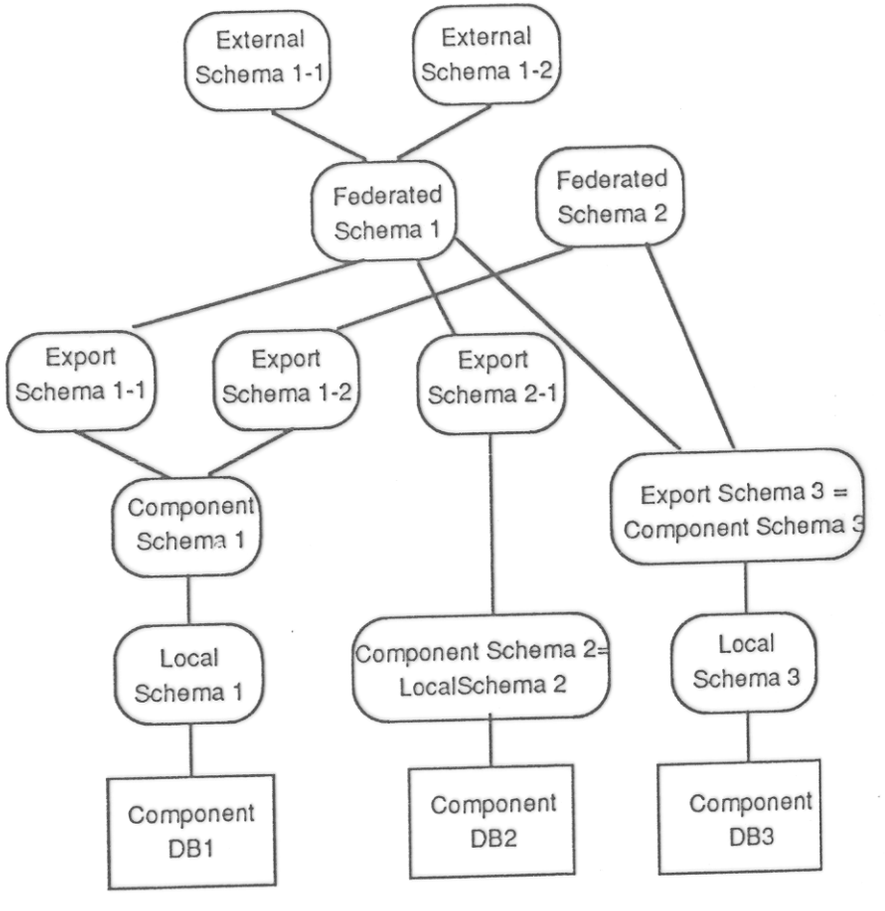
[Sheth and Larson 90]

Five-Level Schema Architecture of a FDBS



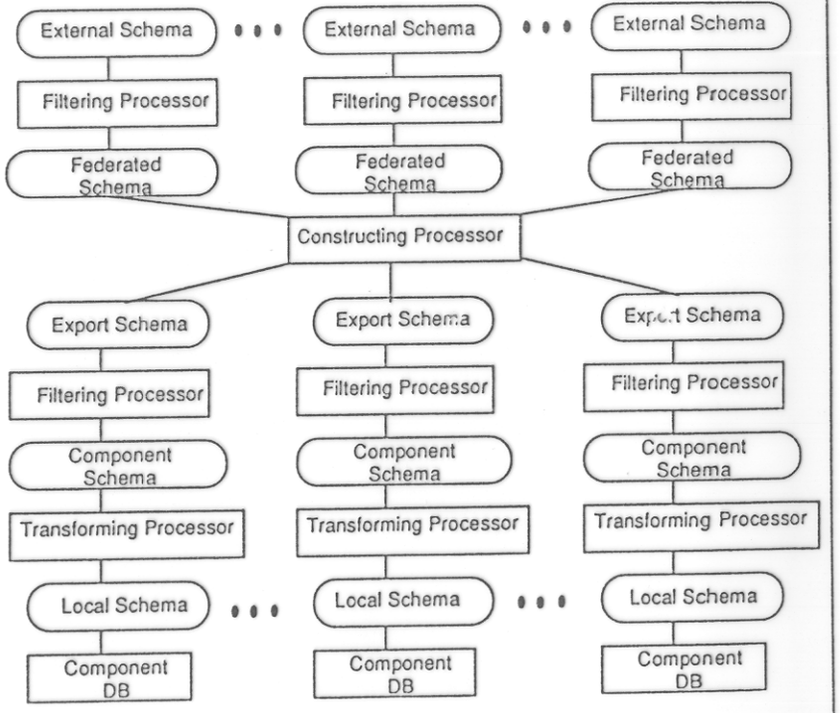
[Sheth 88c]

An Example FDBS Schemas with missing schemas at some levels



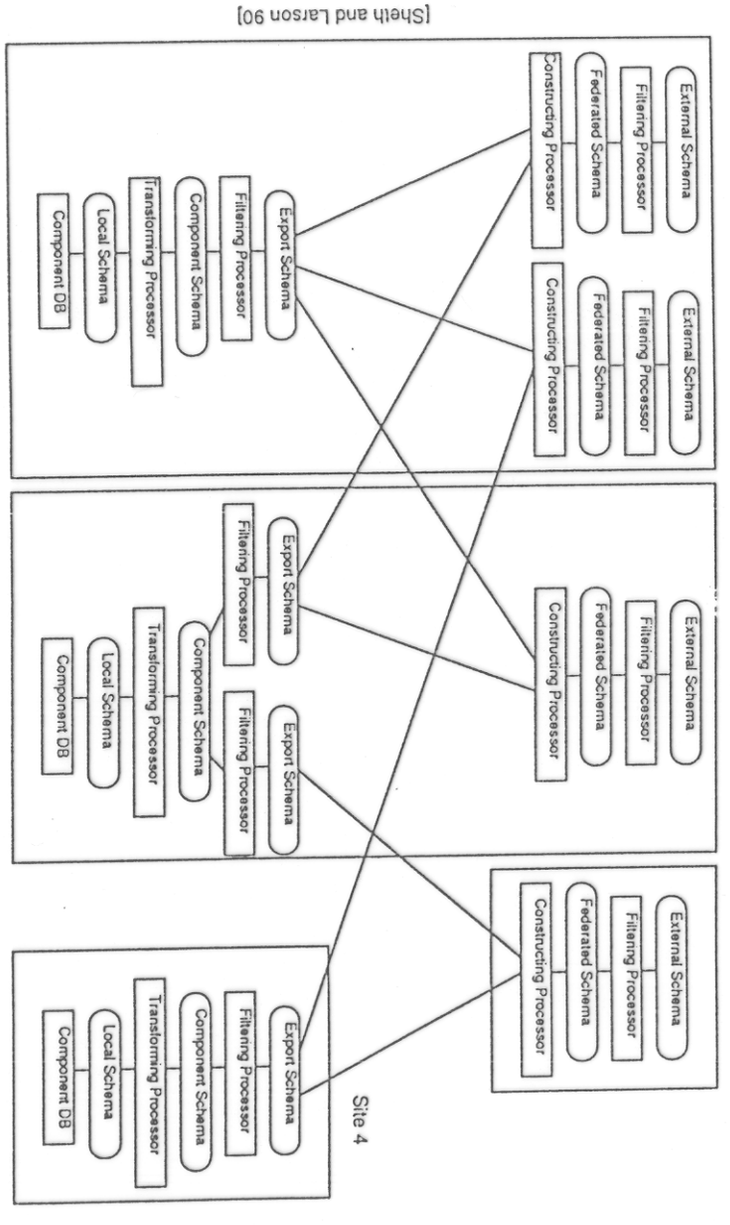
[Sheth and Larson 90]

System Architecture of a FDBS



[Sheth and Larson 90]

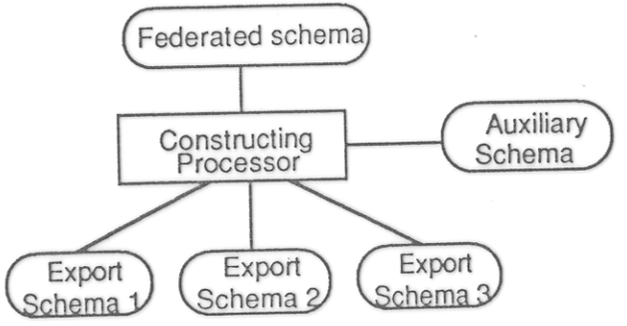
A Typical FDBS System Configuration



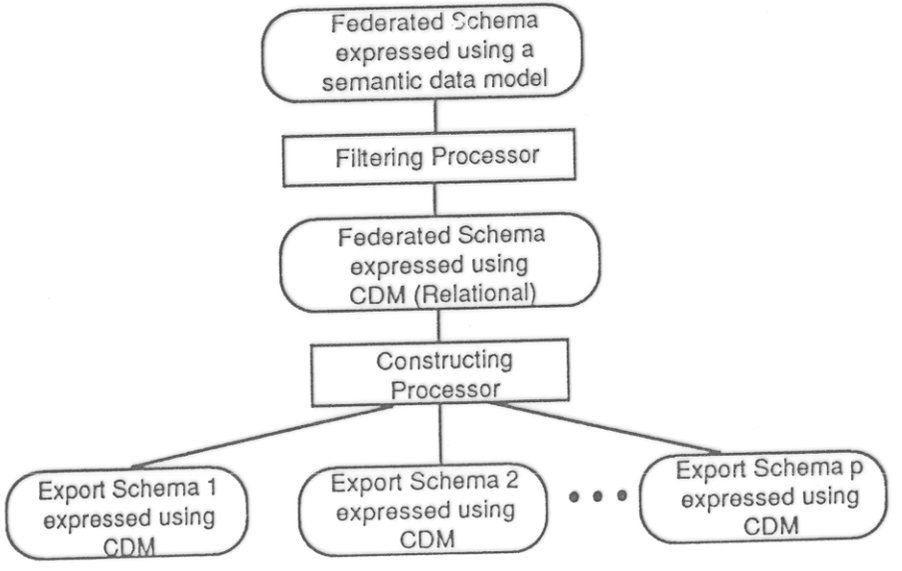
[Sheth and Larson 90]

PROCESSOR USE EXAMPLES

Using an Auxiliary Schema to Store Translation Information Needed by a Constructing Processor

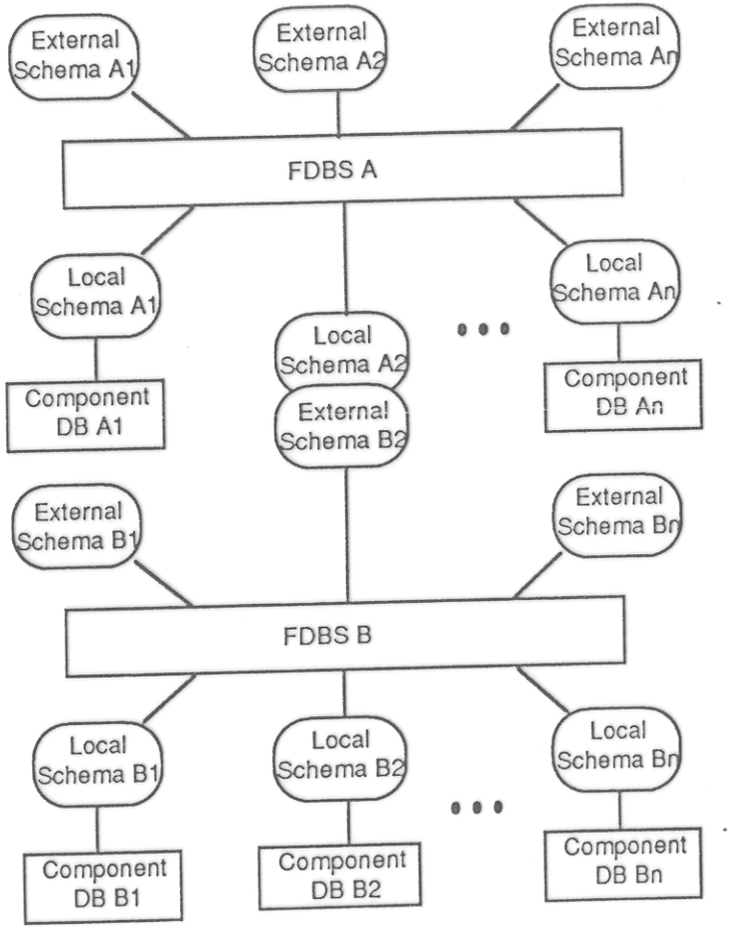


Using a Filtering Processor to Enforce Constraints Across Export Schemas



[Sheth and Larson 90]

FDBS B acting as a Backend to FDBS A



[Sheth and Larson 90]

METHODOLOGY

Three Phases

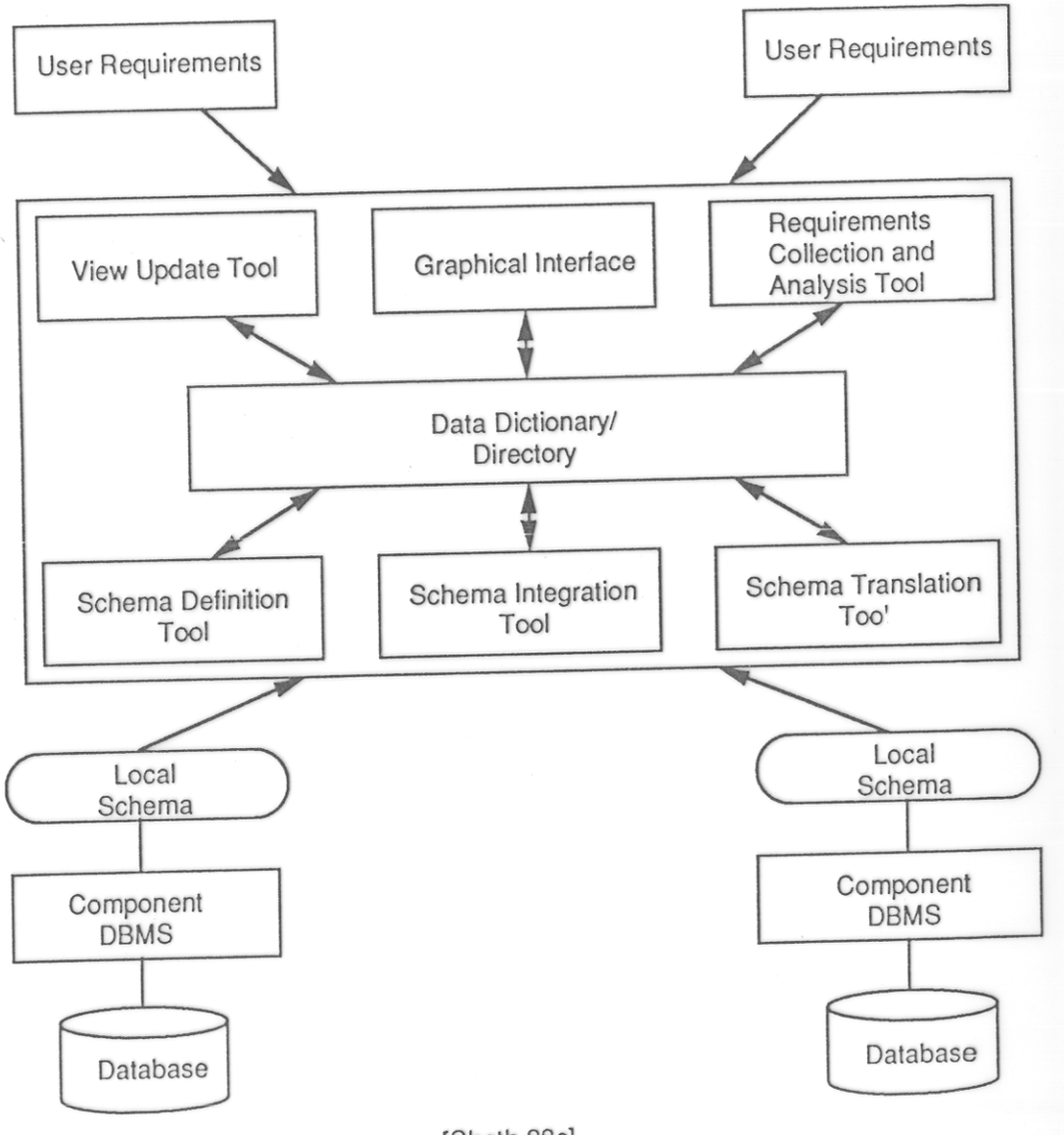
1. Preintegration
   - reverse engineering

2. Design and Integration
   - schema translation
   - schema integration
   - negotiation
   - (access control)
   - (view update)
   - database design

3. Operation
   - multidatabase language
   - operation transformation
   - query processing and optimization
   - transaction management
     -- concurrency control and recovery
   - (access control)/security
   - (view update)

These phases need not be performed serially.
Multiple iterations are often needed.

Tools for Developing FDBSs

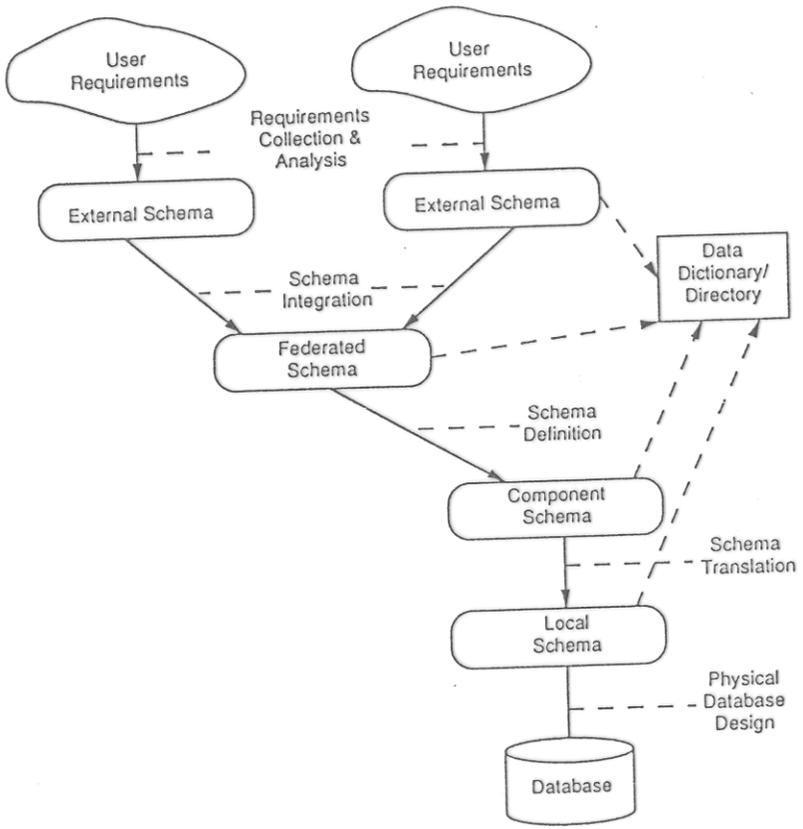


[Sheth 88c]

Sheth 88

METHODOLOGY

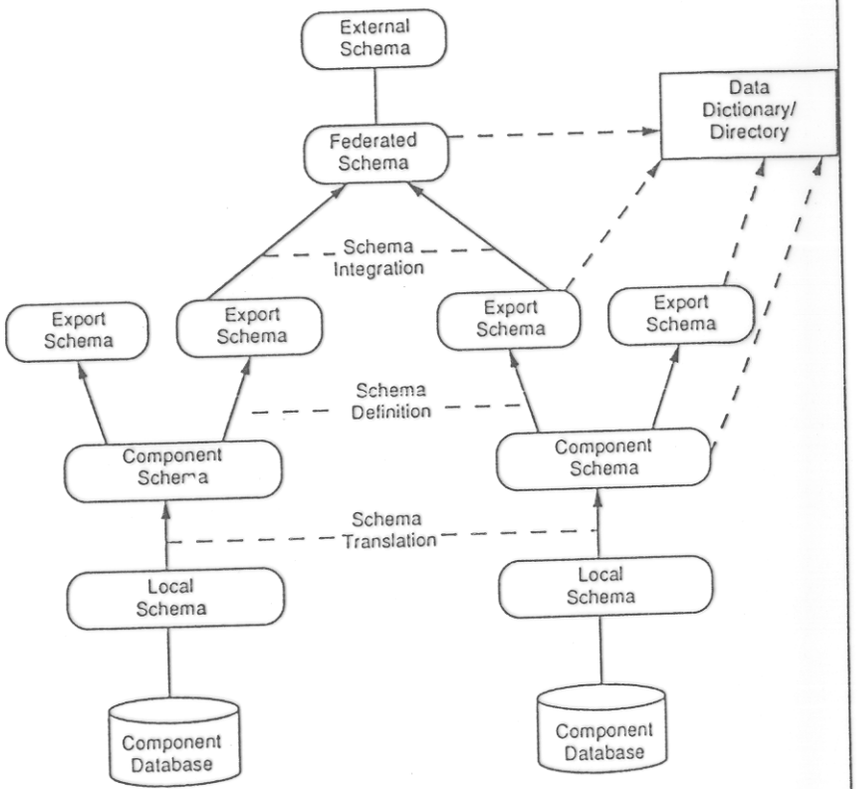
Top-Down FDBS Building Process



[Sheth 88c]

METHODOLOGY

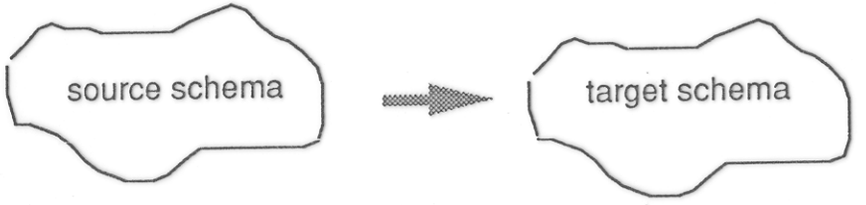
Bottom-Up FDBS Building Process



[Sheth 88c]

3. SCHEMA TRANSLATION

Problem: Given a source schema in one model, map it into an equivalent target schema in another model. If the source schema and target schema are according to the same data model, we have a schema restructuring mapping, otherwise we have a schema translation mapping.



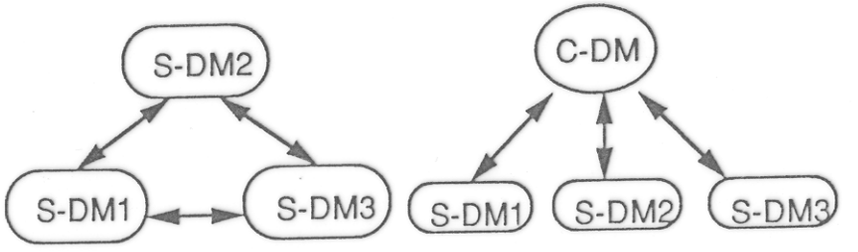
Two schemas are equivalent if
(1) they describe the same database,
(2) any command expressed in terms of the object of one schema can be translated into commands expressed in terms of the objects of the other schema, and
(3) the same changes are made to the database if either source command or the target commands are executed.

[Larson 83] [Tsichritzis and Lochovsky 82]

Two ways to perform translation:
Specify mapping between every pair of data models OR use a common (canonical/mapping) data model

Examples of common data models used in various schema translation efforts are:

Relational [Klug 80] [Breitbart et al 84]
Binary [Pelagatti 78] DIAM [Senko 76]
Extended Set Theory [Sibley and Hardgrave 77]
E-R type [Katz 80] [Elmasri et al 85]
Funcational [Landers and Rosenberg 82]



S-DMi = Schema in Data Model "i"
C-DM = Schema in Common Data Model

SCHEMA TRANSLATION
Comparision Summary of Mapping:
Representation Structures of three models

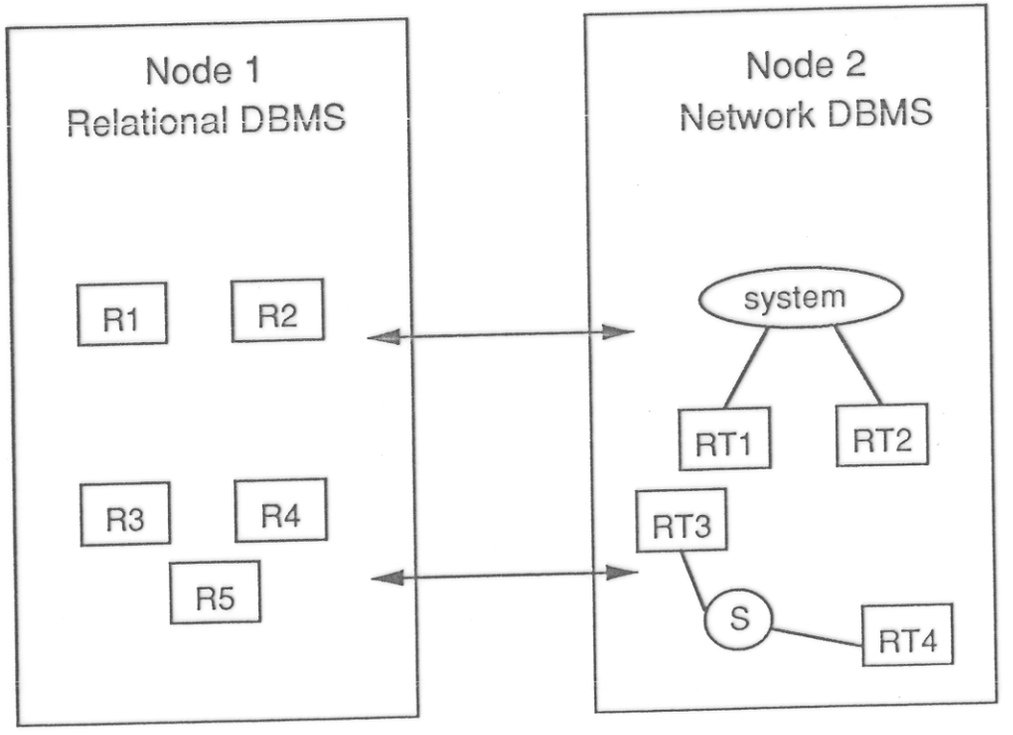
ECR	Relational	Network (DBTG)
Entity-type	Relation	Record type
Category (ISA)	Relation	Record type and single owner single member set
Category(generalization)	Relation	Record type and single owner multiple member set
Functional relationship (1:1,!:N)	Foreign key	Single owner single member set
Non-functional relationship (M:N)	Relation	Record type and owner-member sets
Single valued atribute	Attribute	Field
Multi-valued attribute	Attribute 'Nest' relation	Repeating group, vector
Entity identifier	System assigned attribute	Database key

[Elamsri et al 85]

NETWORK TO RELATIONAL TRANSLATION

Structural Similarities:

Network Model	**Relational Model**
Record type	Relation
Data item of the record	Column of the relation
Record key	Relation Key
Set	Foreign Key or Relation



Some aspects of Network Model deserve careful examination:

Set Participation: Foreign Key or New Relation

Set Ordering: New Column

Repeating Groups:
Restructuring network schema or repeating values in a column

System-owned set: New column

Areas: New column

[Larson 83]

Network to Relational Translation (continued)

(1)For each record type Ni define a relation Ri such that:
(a) Ri contains one attribute for each data item of Ni.
(b) If Ni has a key, the key of Ri is equal to the key of Nj; otherwise, the key of Ri is equal to the database key of Ni, which appears as an explicit attribute of Ri.

(2) For each link Lij, with owner record type Ni and member record type Nj, define relational constraints and change the existing relations such that:
(a) The key of Ni appears as a foreign key of Rj.
(b) One of the following sets of constraints applies, depending on the type of set membership (Ri and Rj are the relations corresponding to Ni and Nj, respectively).
- Fixed Automatic
- Mandatory Automatic
- Optional Automatic
- Fixed Manual
- Mandatory Manual
- Optional Manual

(c) Instead of explicit constraints of (b), it is possible to derive relational schema that contains only relations and multivalued dependencies if network schema is loop-free and the links are fixed automatic.

[Tsichritzis and Lochovsky 82]
[Vassiliou and Lochovsky 80] [Lien 80]

SCHEMA TRANSLATION Remarks

* Easier Translations:

  Procedural to Non-procedural
  Conceptual to Representation

  E.g. Network to Relational
  ECR to Relational

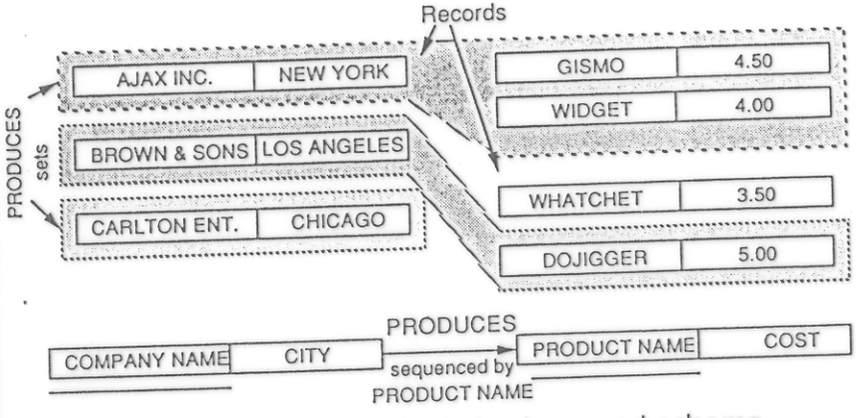
* More Difficult Translations:

  Non-procedural to Procedural
  Representation to Conceptual

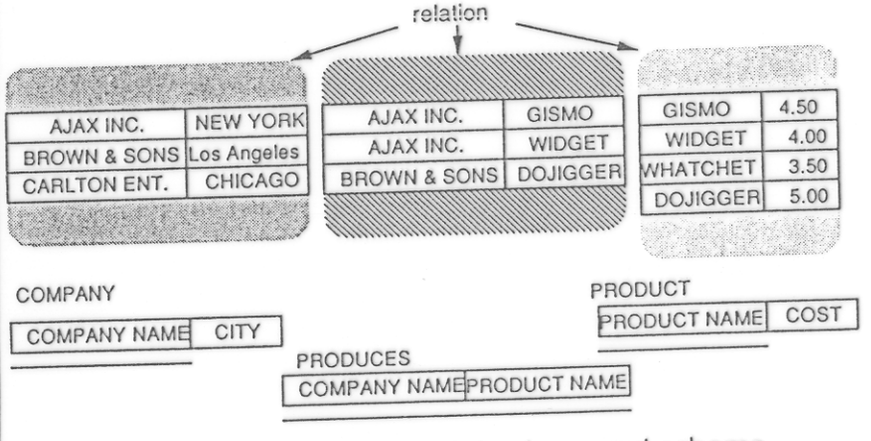
  E.g. Relational to Network
  Relational to ECR

[Tsichritzis and Lochovsky 82]

Network to Relational Translation
An Example



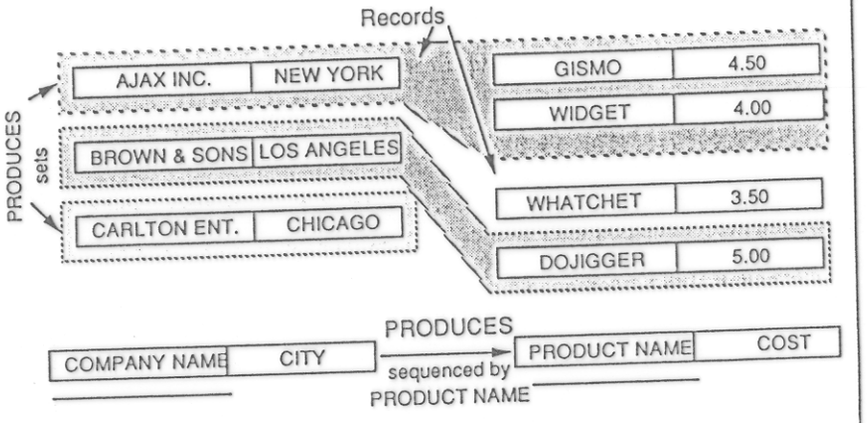
Source schema: A network database and schema .



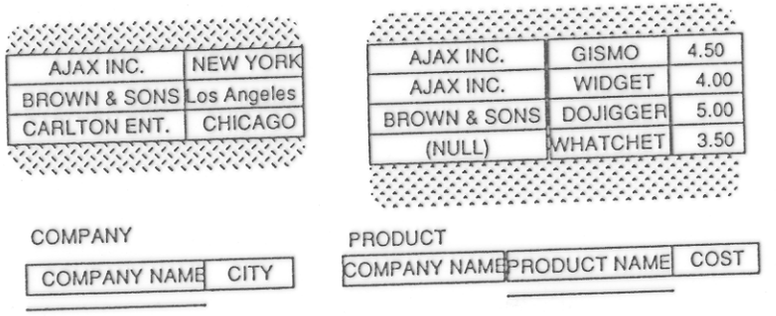
Target schema: A relational database and schema.

[Larson 83]

Network to Relational Translation
An Example: Alternate Translation



Source schema: A network database and schema .



Target schema: A relational database and schema.

[Larson 83]

Network to Relational Translation
An Example: Possible Mapping

1. R-COMPANY.Company Name <- N-COMPANY.Company-Name
   R-COMPANY.City <- N-COMPANY.City
   R-PRODUCT.Product-Name <- N-PRODUCT.Product-Name
   R-PRODUCT.Cost <- N-PRODUCT.Cost

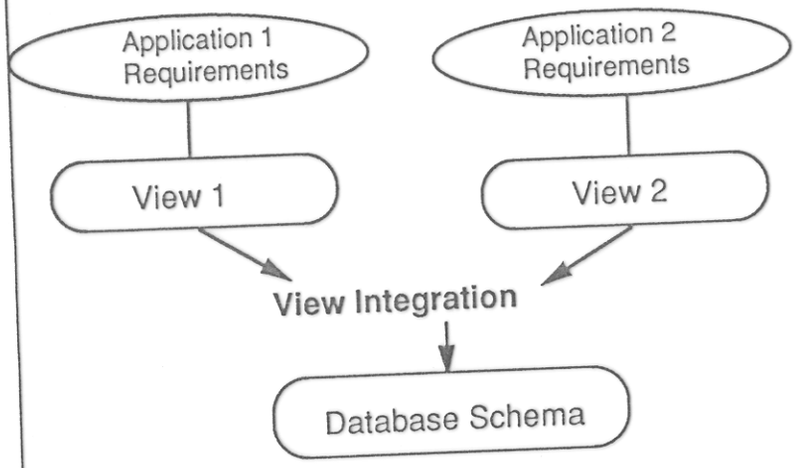
   Key(R-PRODUCT) = Product-Name
   <- Key(N-PRODUCT) = Product-Name

2. Foreign-Key(R-PRODUCT.Company-Name)
   <- Key(R-COMPANY.Company-Name)

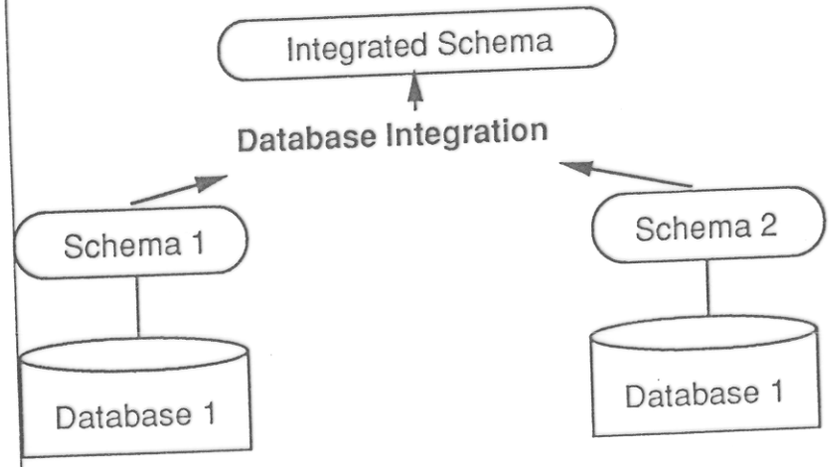
   Explicit Constraints = Optional Automatic
   R-PRODUCT.Product-Name sequenced

SCHEMA INTEGRATION

Logical Database Design:
(*top-down design*)



Database Integration:
(*bottom-up design*)



Identify relationships (dependency) among objects in different schemas/databases

Develop corporate wide view of logical data

-- identify redundancy and remove or maintain
-- increase sharing of data, provide uniform access
-- maintain consistency of related data
-- provide better understanding of operations

Schema Integration Activities

Preintegration:

- All schemas should be in the same data model;
- Choice of Common Data Model is critical
- Specify global constraints and naming conventions

Comparing:

- Also called Schema Analysis
- Identify schema components that represent the same information/semantics
- Define Attribute ~~Equivalence~~ Relationship
- Assert Object (Entity/Relatioship) Equivalences

Conforming:

- Same information should be represented using the same model constructs

Merging:

- Components from different schemas are merged to create an integrated schema; this task can be automated

Restructuring:

- Integrator can use operators to make changes in the integrated schema to capture semantics not used earlier in the process

[Batini et al 86]

Resolving Incompatabilities

Types of Incompatibilities:

1. Differences in Models
2. Differences in Naming Convensions
3. Differences in Representations
4. Differences in Scales/Units
5. Missing Data
6. Conflicting Data Values

Solutions

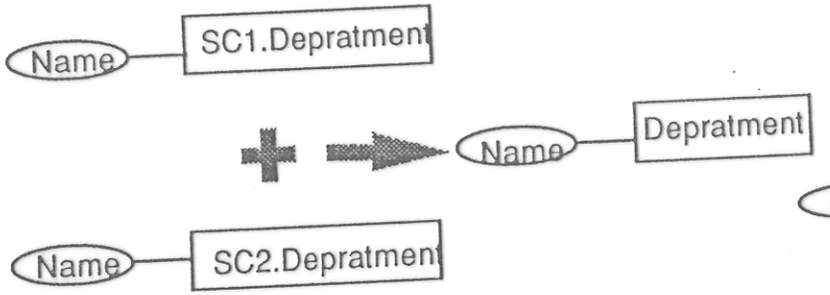
A. Schema Translation to solve 1.
B. Schema Analysis and by
   - Renaming
   - Logical Restructuring
   - Scale Conversion

   These enable us to solve 2 thru 4.
B. Auxiliary Database can be stored with auxiliary schema attached to the unified schema to slove 5 and 6.

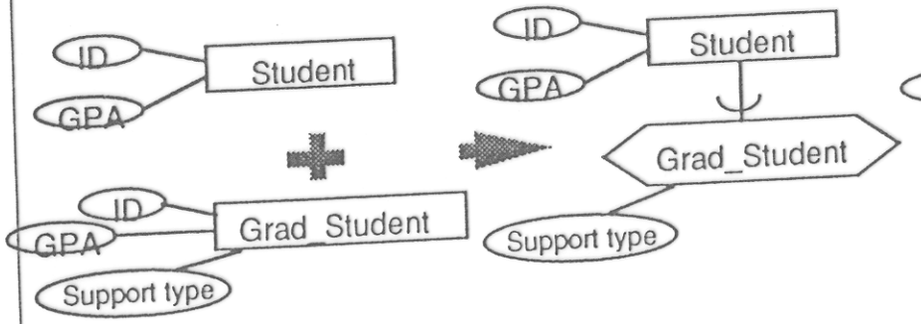
[Landers and Rosenberg 82] [Litwin Abdellatif 86]

SCHEMA INTEGRATION
Object Relatioships

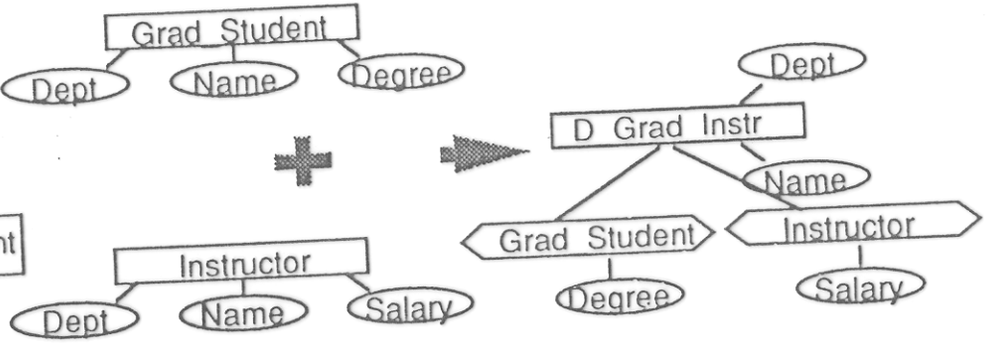
Case 1: Identical Domains



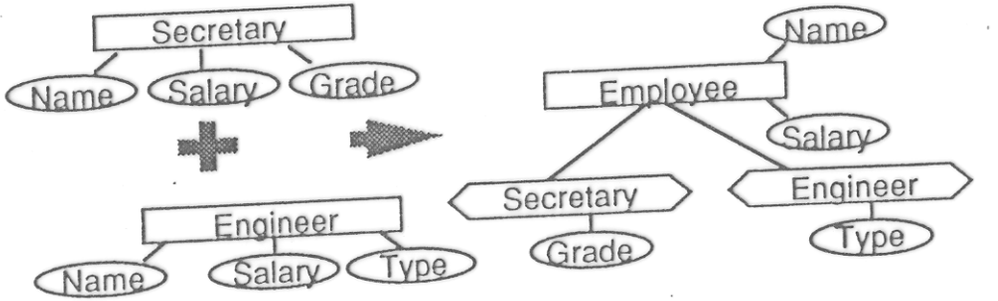
Case 2: Contained Domains



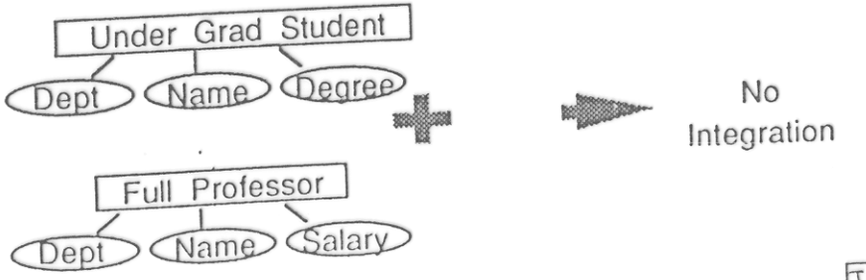
Case 3: Overlapping Domains



Case 4: Disjoint Domains



Case 5: Non-integrable Domains



[Navathe et al 86]

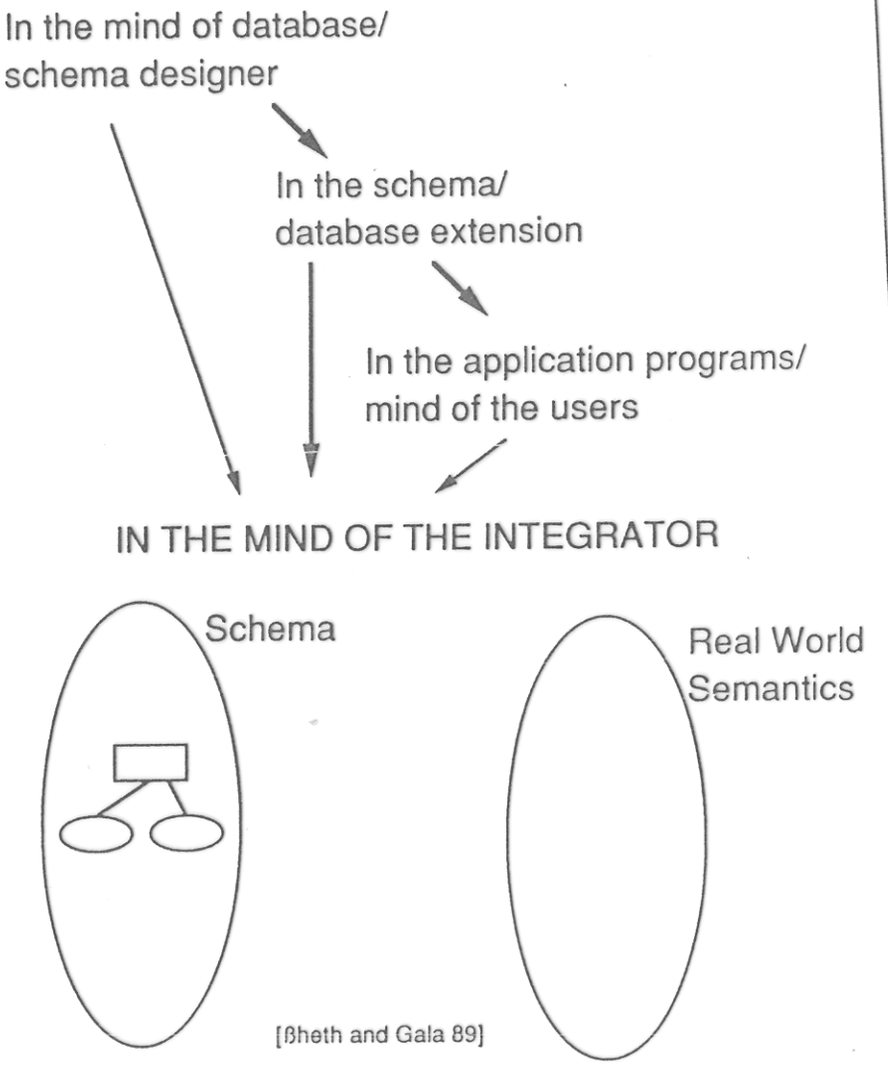
SCHEMA INTEGRATION: Observations

The key problem in integrating schemas is our **inability to completely capture the semantics** and being able to argue/reason about the semantics.

We can perform some automated reasoning about the object relationships, but **reasoning about attribute relationships cannot be automated**, it has to be done by a human.

There is no unique integrated schema for a given set of component schemas. Different semantics given by the integrator will result in different integrated schemas.

WHERE IS SEMANTICS?



[ßheth and Gala 89]

A Case Study on Schema modeling, Analysis, and Integration

BERDI

(Bellcore Schema Design and Integration Toolkit)

A prototype software system to develop, evaluate and demonstrate methodology and techniques to

- real world functionality
- real world scale

Some related work:
Tools: Sheth et al 88; Hayes & Ram 90; Sheth & Marcus 92
Techniques & Concepts: Batini et al 86; Batini et al 91; Collet et al 91; Dayal & Hwang 84; Elmasri, et al 86; Navathe & Gadgil 82; Spaccapietra et al 91, ...

Solving the right problem: A perspective on Schema Integration

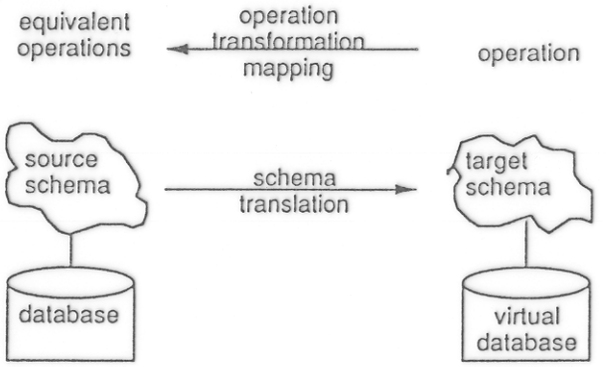
Different perspectives, different needs:

- Bottom-up (schema) integration-- To develop integrated schema(s) to give transparency to the distributed/heterogeneous databases?
- Top-down (view) integration --To integrate views/subschemas of the same application to develop the database schema for that application?
- To analyze schemas for dependencies among them to facilitate interoperability?
- To develop (the schema of) a new application that encompasses one or more existing applications?
- To perform a ***shallow*** analysis/integration (e.g., without attributes, functions, or behavior), and/or ***deep*** analysis/integration resulting in complete dependency/integration mapping?

[Sheth 91a]

OPERATION TRANSFORMATION

Given a source schema, a database according to the source schema and an equivalent target schema, map operations on target schema to equivalent operations on source schema. If the source and target schemas are according to the same data model then we have view mapping, otherwise operation transformation mapping. Operation transformation mapping is used to translate a query operation on a target schema into equivalent operations on the source schema.



OPERATION TRANSFORMATION

Query Translation Relational to Network

Transforming nonprocedural relational data manipulation command into a sequence of procedural network commands:

Select =>
A looping statement in network data manipulation language in which each record is retrieved and examined. Records satisfying the selection condition are moved to the target relation.

```
T = Select (R1,condition) => r = get_first (N1);
                             Loop: if condition(r);insert r
into T
                             r = get_next(N1); go to
Loop
```

Project =>
A looping statement in which each record is retrieved and the specified items are copied to the target table.

Join =>
Two nested loops in which the outer loop reads successively records from one record type and the inner loop successively reads records from second record type in the set owned by each of the record of the first record type.

OPERATION TRANSFORMATION

View Mapping
(through Query Modification)

DATABASE RELATIONS:

EMPLOYEE

NAME	ADDRESS	DEPT
Smith	15 Bloor	Paint
Jones	25 King	Toy
Carter	171 Yonge	Paint

DEPARTMENT

DEPT	LOCATION	MANAGER
Paint	King	Miles
Toy	Queen	Quin
Pipe	King	Miles

VIEW DEFINITION:

```
EMPVIEW <-
 SELECT
EMPLOYEE.NAME,ADDRESS,MANAGER
 FROM EMPLOYEE,DEPARTMENT
 WHERE EMPLOYEE.DEPT=DEPARTMENT.DEPT
```

VIEW:

EMPVIEW

NAME	ADDRESS	MANAGER
Smith	15 Bloor	Miles
Jones	25 King	Quin
Carter	171 Yonge	Miles

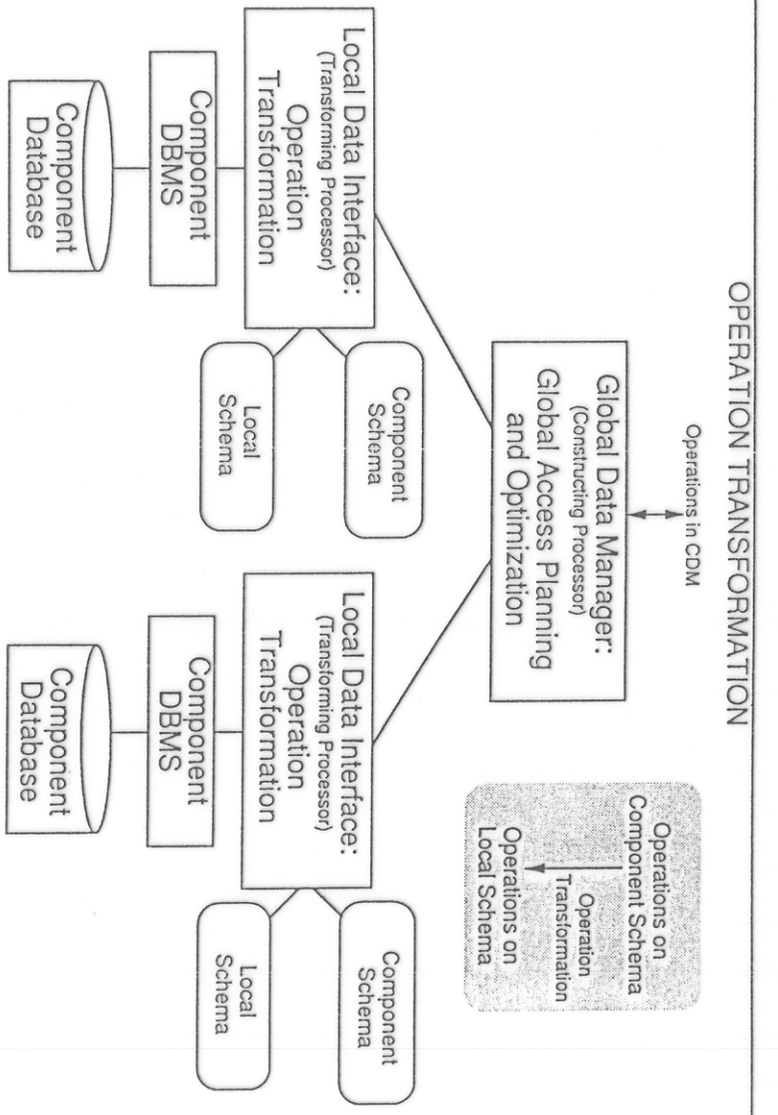
Now a query on EMPVIEW:

```
ANSWER <-
  SELECT MANAGER
  FROM EMPVIEW
  WHERE NAME = 'Smith'
```

can be rewritten into the following query by a technique called Query Modification:

```
ANSWER <-
  SELECT MANAGER
  FROM DEPARTMENT
   WHERE DEPARTMENT.DEPT = SELECT
EMPLOYEE.DEPT
                          FROM EMPLOYEE
                          WHERE NAME = 'Smith'
```

[Stonebraker 75] [Tsichritzis and Lochovsky 82]



Query Translation
Relational to Network

Transforming nonprocedural relational data manipulation command into a sequence of procedural network commands:

Select =>
A looping statement in network data manipulation language in which each record is retrieved and examined. Records satisfying the selection condition are moved to the target relation.

```
T = Select (R1,condition) => r = get_first (N1);
                             Loop: if condition(r);insert r into T
                             r = get_next(N1); go to Loop
```

Project =>
A looping statement in which each record is retrieved and the specified items are copied to the target table.

Join =>
Two nested loops in which the outer loop reads successively records from one record type and the inner loop successively reads records from second record type in the set owned by each of the record of the first record type.

OPERATION TRANSFORMATION

Query Translation

Remarks

Operation Mapping can be established by direct mappings or by using intermediate mapping data model as in schema translation.

Following complexities exist in query translation:

1. Translating procedural operations into nonprocedural operations is very difficult. (This may be an important reason for not choosing a procedural model as a representation model.)
2. Dynamic creation of tables is required.
3. Translating conjuctive (OR) terms in a relational query is difficult.

[Larson 83] [Vassiliou and Lochovsky 80]

6. QUERY PROCESSING AND OPTIMIZATION

Query Processing

Goal: Utilizing the potential for parallel processing in a distributed system and minimizing processing cost by reducing the amount of data moved between sites.

Issues: Additional complexities introduced by the heterogeneous environment.

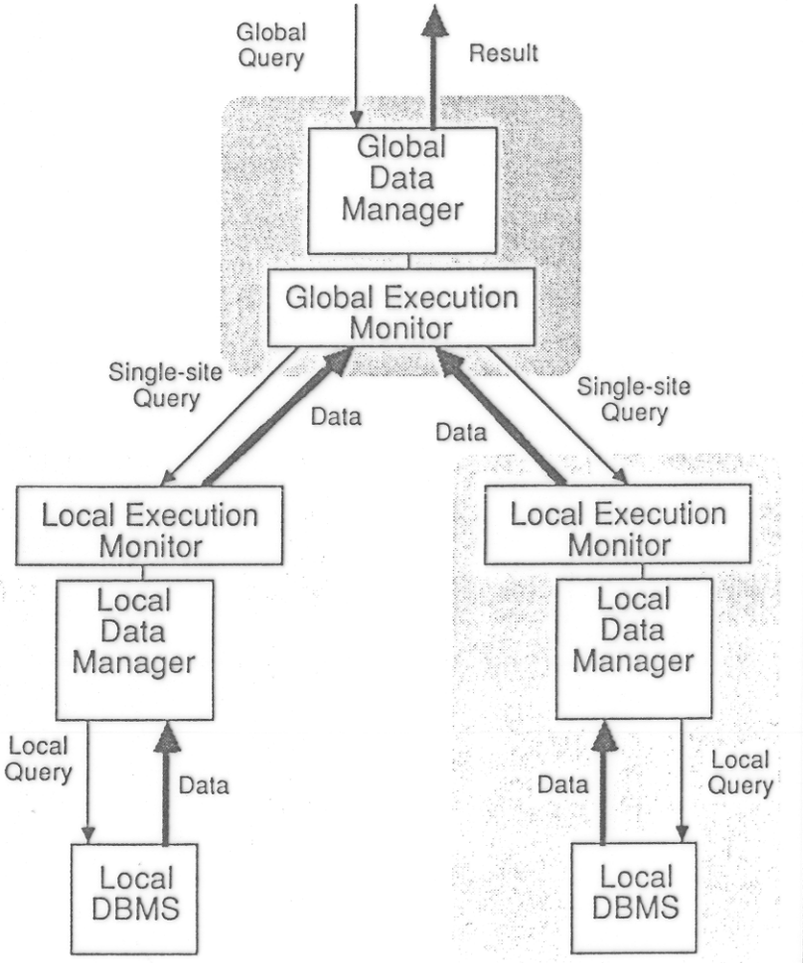
(1) Cost of performing a local query may differ greatly from site to site.

(2) The ability to recieve and reference "moved" data may vary from site to site. Many DBMS do not support creation and loading of temporary databases.

(3) Local querying capabilities may be limited.

(4) Local DBMSs vary in their capabalities to optimize.

[Landers and Rosenberg 82]



Global Optimization

There is a trade-off between the amount of work done by the GDM and the software complexity of LDI. There is also a trade-off between amount of communication and processing done by different sites. Depending on how a global transaction is divided, a range of alternatives is possible.

Examples of alternatives are:

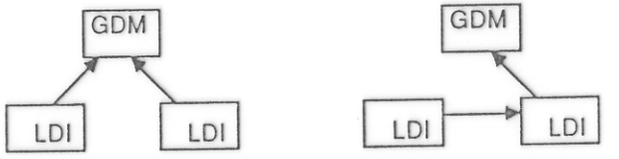
(1) More work by GDM, more communication but simple: Translate the global query in smallest possible single site subqueries. Each site may be sent several subqueries. Collect partial result at GDB and merge.

(2) Medium work by GDM, medium communication, medium complexity of GDM and LDI's: Translate the global transaction into largest possible single site subtransactions (one transaction for each site where data is located). This will reduce communication and needs less work when merging partial results. However, the LDI's have to be sophisticated.

Global Optimization

(3) Less work by GDM, less communication, complex GDM and LDI's: Generate efficient programs such that LDI's participate in global optimization. Partial results can be sent to GDM or other LDI's. LDI's have to be able to support some additional functionalities like sorting, removing duplicates, handling and merging temporary files.

E.g., consider multisite join:



Local Optimization

Goal: Optimize processing of the subtransaction received from GDM against the local DBMS.

Opportunities for Optimization:

- Local Query Language
- Physical Database Organization
- Access Path Selection

[Onuegbe et al 83] [Dayal and Goodman 82]

Local Optimization

Sequences of joins, selects, and projects can be optimized.

E.g., a select followed by a project can be combined into a single loop that selects records satisfying the select condition and copies the projected items.

TRANSACTION MANAGEMENT AND CONCURRENCY CONTROL

A transaction is a set of operations on database. A traditional transaction has following ACID properties:

1. ATOMICITY: Either all or none of the operations should "happen". It commits or aborts.

2. CONSISTENCY: The transaction should be a correct state transformation. This is typically left to the programmer.

3. ISOLATION: Each transaction should see a correct picture of the database state, even if concurrent transactions are executing.

4. DURABILITY: Once a transaction commits, all its effects must be preserved, even if there is a failure.

Serializability is the well known criteria used to define "correct" executions of concurrent transaction.
Current research seeks to address the following question: Which of these properties are appropriate or required in a FDBS and how can they be selectively relaxed? What are useful transaction models? How do autnomy and heterogeneity affect transaction management?

[Haeder and Reuter 83]

CONSISTENCY CRITERIA

Local Consistency	Single Database
Global Consistency	Distributed Databases-No Redundancy
Mutual Consistency	Distributed Databases with Redundancy

Local and Global Consistency:
Conflict Serializability

Weak Serializability: final state serializability, view serializability, quasi serializability

Mutual Consistency:
Single Copy Serializability
Eventual Serializability

CONCURRENCY CONTROL

There are three methods of Concurrency Control:

* Locking
* Timestamping
* Optimistic

Locking: If a transaction accesses a data item, it sets a lock on it. Another transaction with a conflicting lock request has to wait until the lock is released.

Timestamp: Each transaction get a timestamp at the begining and the conflicting transactions are processed according to the timestamp.

Optimistic: After a transaction is executed, it checks if it could have conflicted with other transaction. If so it "aborts" otherwise it "commits".

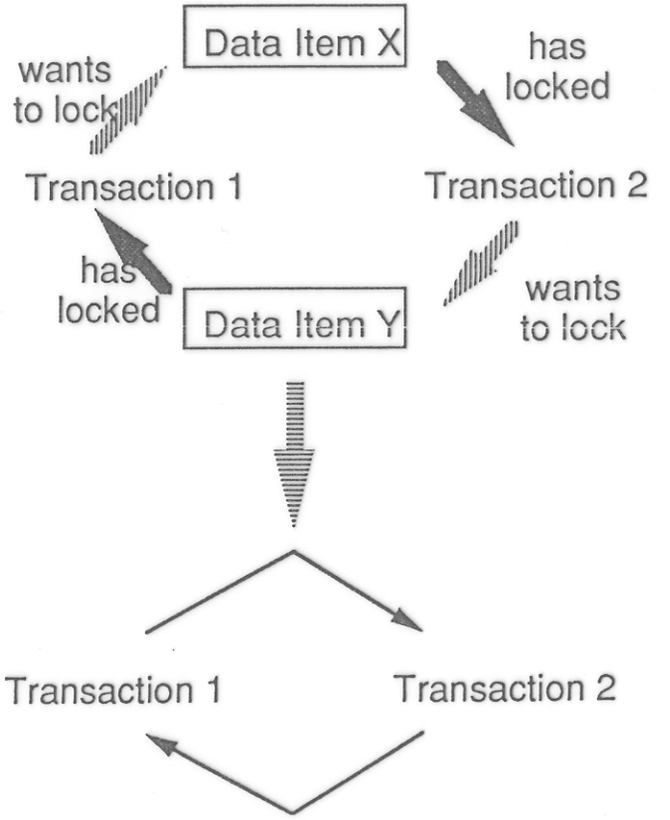
Two transactions conflict if one of them write a data item while other reads or writes the same data item.

[The above definitions are incomplete and oversimplified.]

[Bernstein and Goodman 81]

CONCURRENCY CONTROL

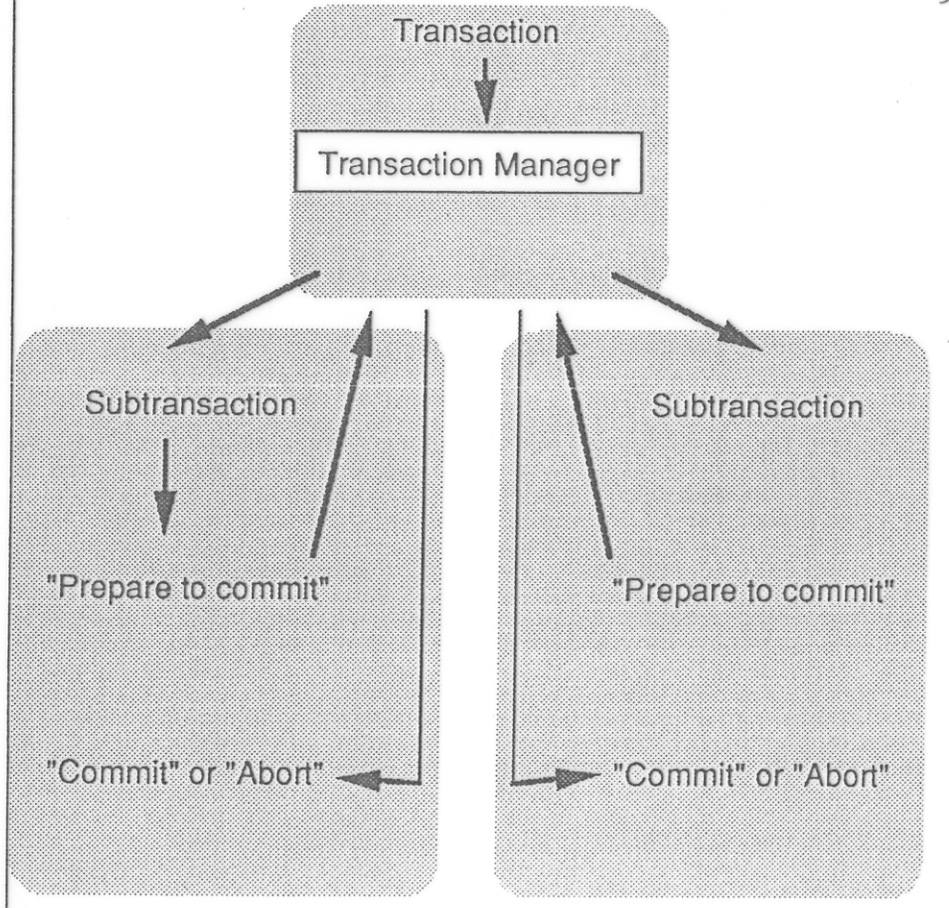
Locking is the most popular method. But, whenever a locking is used, deadlock can occur.



Arrows show wait-for graph. A cycle in a wait-for graph implies a deadlock.

TRANSACTION MANAGEMENT

In order to preserve atomicity property of the transaction, TWO-PHASE COMMIT protocol is used.



[Bernstein and Goodman 81]

EFFECT OF HETEROGENEITY

1. Different DBMSs use different Concurreny Control methods. E.g., if the global TM uses a two phase locking mechanism and a component DBMS uses a timestamp ordering, the latter cannot provide the lock point information required by the former.

2. Even if the Concurrency Control methods are the same, implementations may vary significantly. E.g., locking methods may vary in deadlock handling methods, number of items locked by one lock (locking granularity) may vary, etc.

3. The DBMSs may vary in commit protocols. E.g., Some may not provide two phase commit, while some may.

[Logar and Sheth 86]

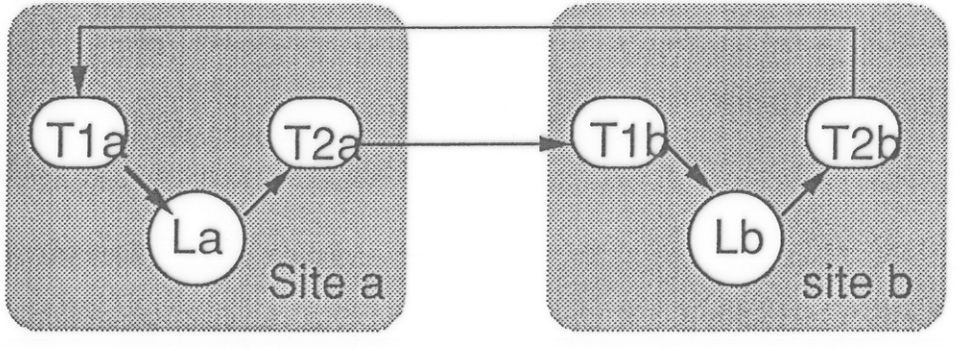
EFFECT OF AUTONOMY

Global Deadlock Detection Problem

A global deadlock involves transactions at more than one site. Problem of global deadlock detection is very difficult because (1) a local process (part of local DBMS or Operating System) does not know about non-local transactions, and (2) a global process does not know about the local transactions.

Most practical solution seems to be time-out.

Example of global deadlock:



[Logar and Sheth 86]

EFFECT OF AUTONOMY

1. Design Autonomy leads to all problems listed in "Effect of Heterogeneity".

2. Even if a component DBMS has the information required for global TM, communication autonomy may prevent the global TM from accessing it.

3. If the global TM has detected a conflict, execution autonomy of a component DBMS may prevent it from taking actions (e.g., block or restart a transaction) necessary to resolve the conflict.

[Du et al 89b]

EFFECT OF AUTONOMY

Direct Conflict vs Indirect Conflict

Consider
Component Databases/DBMSs:: D_1: a; D_2 : b,c
Global Transactions:: G_1:$W_{G1}(a)R_{G1}(c)$
G_2:$R_{G2}(a)W_{G2}(b)$
Local Tansaction at D_2:: L: $R_L(b)W_L(c)$
Local Execution E_1 at D_1:: $W_{G1}(a)R_{G2}(a)$
Local Execution E_2 at D_2:: $W_{G2}(b)R_L(b)W_L(c)R_{G1}(c)$

In E_1, G_1 and G_2 directly conflict,
in E_2, G_1 and G_2 indirectly conflict through the operations of L.

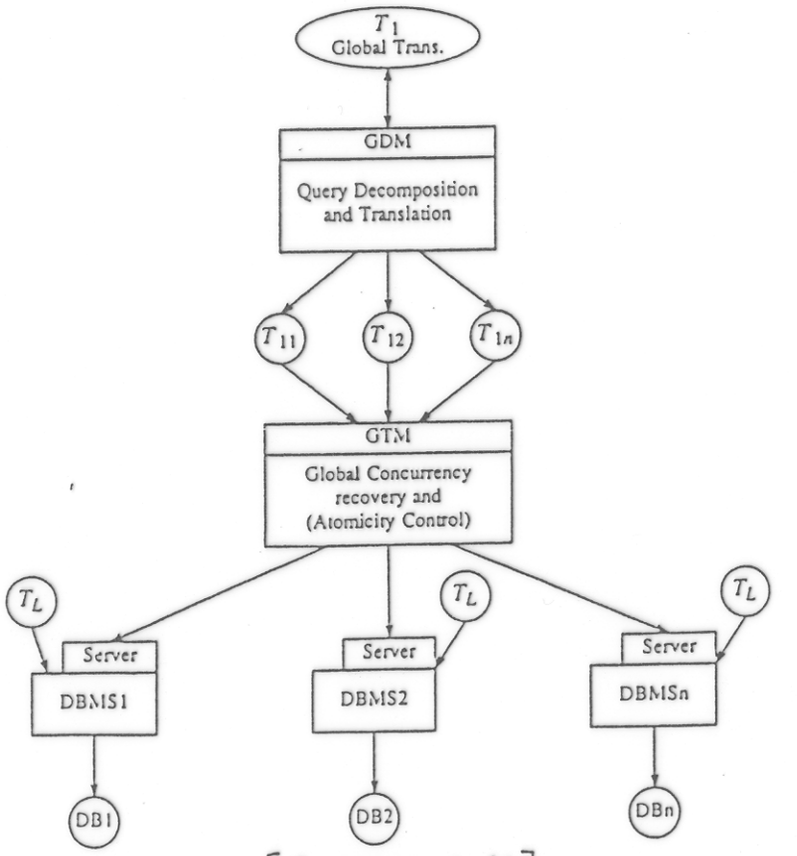
Execution Order vs Serialization Order

The serialization order between two operations might be different than their execution order. The indirect serialization order of $R_{G1}(c)$ and $W_{G2}(b)$ in E_2 remains unchanged even is the execution order is change to E_3.

E_2:: $W_{G2}(b)R_L(b)W_L(c)R_{G1}(c)$
E_3:: $W_L(c)R_{G1}(c)W_{G2}(b)R_L(b)$

[Du et al 89]

The Transaction Processing "Architecture" for HDBMS



[ELMAGARMID 88]
[Lugar & Sheth 86]

TRANSACTION MANAGEMENT AND CONCURRENCY CONTROL

Alternatives

1. Allow only read (retrieval) operation. Do not allow write (update) operation. Concurrency control problem does not arise if all transactions are read only.

2. Assume all DBMSs provide locking and support two-phase commit. Detect deadlocks by time out.

3. **Limit concurrency.** (e.g.,allow one global transaction per component DBMS; let the locking granularity be single site and check acyclicity of the site graph [Logar & Sheth 86][Alonso et al 87]; limit local transaction to be read only or write only on non-replicated data. [Breitbart and Silberschatz 88]

4. **Compromise autonomy, or require info for global CC, or enforce specific serialization order.** [Sugihara 87]. [Pu 87]? Also DDTS, top-down approach [Elmagarmid & Du 90] to enfore specific serialization order..

5. **Different consistency criteria or relax strict serializability.** E.g., Quasi seriazibility [Du and Elmagarmid 89] (allows only serial execution of global subtransactions). S-Transactions [Eliassen & Veijalainen 87]. Sagas [Garcia-Molina & Salem 87]. Value date [Litwin & Tirri 88]. Flexible transaction [Elmagarmid et al 90].

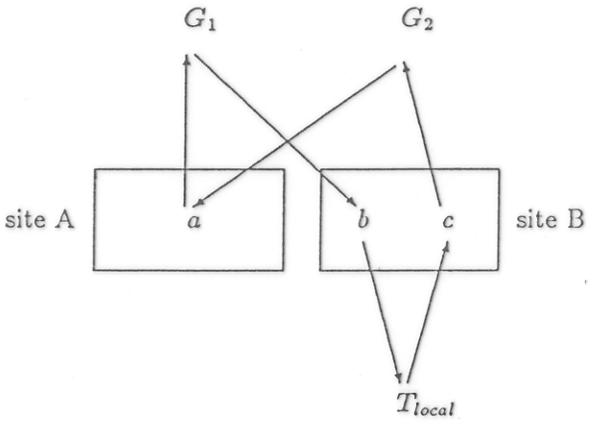
6. **Prevent different schedules of conflicting transactions at different sites without loss of autonomy.** Ticketing method [Georgakopoulos et al 91].

Problems in Enforcing Global Serializability

- Transaction execution and serialization orders can be different.
- DBSs do not report the local serialization order of transactions.
- To determine the global serializability, the MDBS must take into account conflicts caused by local transactions.
- Information about local transactions is not available.

Problems in Enforcing Global Serializability

Example: Serial Global Execution & Strict Local Schedules



site A:

$r_{G_1}(a)\ w_{G_2}(a)$

$G_1 \rightarrow G_2$

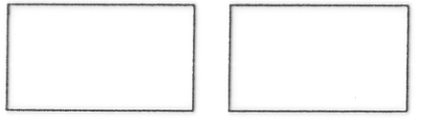
site B:

$w_{G_1}(b)\ r_{G_2}(c)$

Georgakopolous

Problems in Enforcing Global Serializability

Example: Serial Global Execution



$r_{T_{local}}(b)$ $w_{T_{local}}(c)$

$G_2 \rightarrow T_{local} \rightarrow G_1$

Analogous Execution and Serialization Orders

Definition: Let S be a serializable schedule. We say that transactions in S have *analogous execution and serialization orders* if for any pair of transactions T_i and T_j such that T_i is committed before T_j in S, T_i is also serialized before T_j in S.

FACT 1: **Strict schedules do not assure that transactions have analogous execution and serialization orders.**

FACT 2: **Schedulers allowing all schedules in which transactions have analogous execution and serialization orders are expensive, and difficult to design and build.**

Georgakopolous

Rigorous Schedules and Schedulers

Definition *A transaction management mechanism is rigorous if the following two conditions hold: (i) it guarantees strictness and (ii) no data item may be written until the transactions that previously read it commit or abort.*

Theorem: *If a transaction management mechanism ensures rigorousness, it produces (conflict) serializable schedules in which transaction execution and serialization orders are analogous.*

[BGRS91]

Georgakopolous

Ticket Methods

Determining the local serialization order in non-rigorous systems

The subtransactions of global transactions take a *ticket* from the LDBS they access.

The current value of the ticket is stored in the LDBS.

Tickets are logical timestamps. The actions to read ($r(ticket)$) and increment the ticket value ($w(ticket+1)$) are part of each subtransaction.

Theorem: *The tickets obtained by the subtransactions of global transactions determine their relative serialization order.*

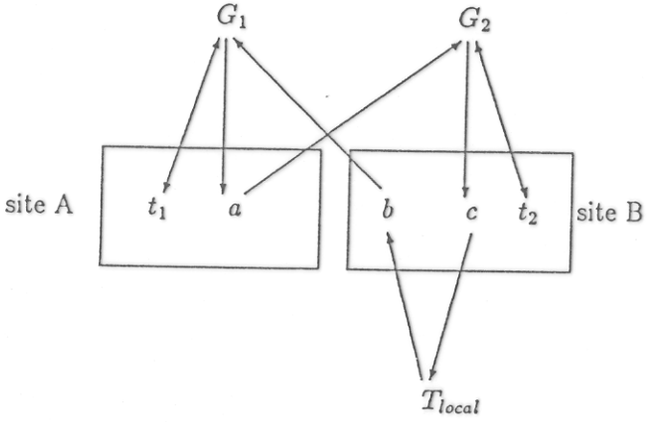
- Tickets create **direct conflicts** between the subtransactions at each LDBS.
- Subtransactions are serialized according to the ticket values they obtain.
- If not, they get aborted by the **local concurrency control.**

[GRS91]

Georgakopolous

The Effects of the Take-A-Ticket Approach

Example



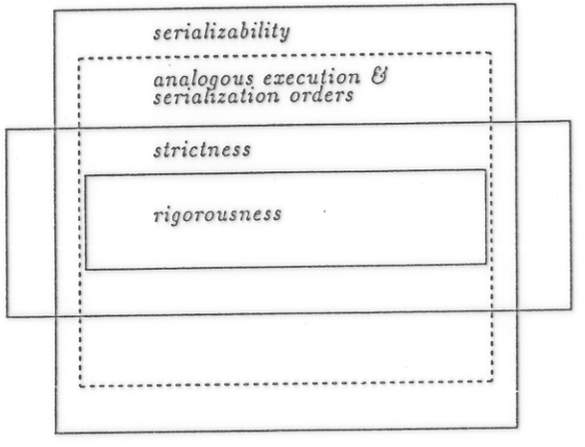
siteA : $r_{G_1}(t_1)w_{G_1}(t_1+1)w_{G_1}(a)r_{G_2}(t_1)w_{G_2}(t_1+1)r_{G_2}(a), i.e.,$
$G_1 \rightarrow G_2$

siteB : $w_{T_{local}}(b)r_{G_1}(t_2)w_{G_1}(t_2+1)r_{G_1}(b)r_{G_2}(t_2)\ w_{G_2}(t_2+1)$
$w_{G_2}(c)r_{T_{local}}(c), i.e.,$
$G_2 \rightarrow T_1 \rightarrow G_1$

[GRS91]

Georgakopolous

Rigorous Schedules and Schedulers
Relation of Rigorousness and strictness



Rigorousness simplifies:

- **Transaction management.**
- **Hierachical composition of autonomous schedulers.**

Class of rigorous schedulers includes:
strict 2PL, conservative TO, and OCC using serial validation

[BGRS91]

Georgakopolous

Rigorous Transaction Management
Example: The The Implicit Ticket Method (ITM)

How ITM determines the local serialization order of each subtransaction?

The commitment order (implicit ticket) of the subtransactions at each DBS determines their relative serialization order.

How ITM enforces global serializability?

For any pair of global transactions G_i and G_j, it is sufficient to commit the subtransactions of G_i before the subtransactions of G_j or vice versa.

Theorem: *ITM guarantees global serializability if the following conditions are satisfied:*

1. *The LDBS are rigorous.*
2. *Each global transaction has at most one subtransaction at each LDBS.*
3. *Each subtransaction has a visible prepare to commit state.*

[GRS91]

Georgakopolous

Using Tickets to Enforce Global Serializability
Example: The Optimistic Ticket Method (OTM)

OTM performs SGT for every global transaction G:

1. **Creates a node for G.**
2. **For every recently committed transaction G_c adds an edge:**

   $G \longrightarrow G_c$ **if at any site** $ticket(G) < ticket(G_c)$, **or**

   $G \longleftarrow G_c$ **if at any site** $ticket(G) > ticket(G_c)$

3. **If there is no cycle, G is validated and is committed.**

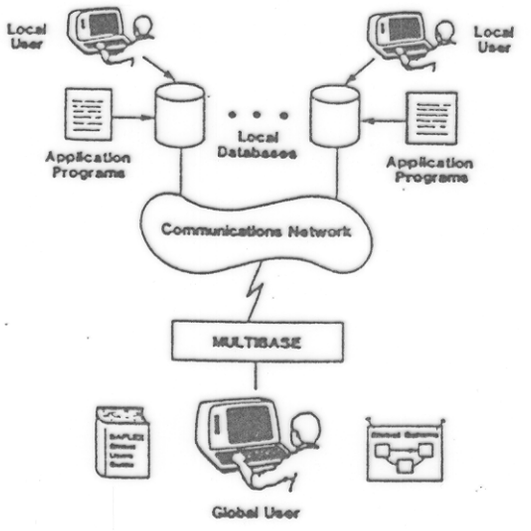
Theorem: *OTM guarantees global serializability if the following conditions are satisfied:*

1. *The LDBS assure local serializability.*
2. *Each global transaction has at most one subtransaction at each LDBS.*
3. *Each subtransaction has a visible prepare to commit state.*

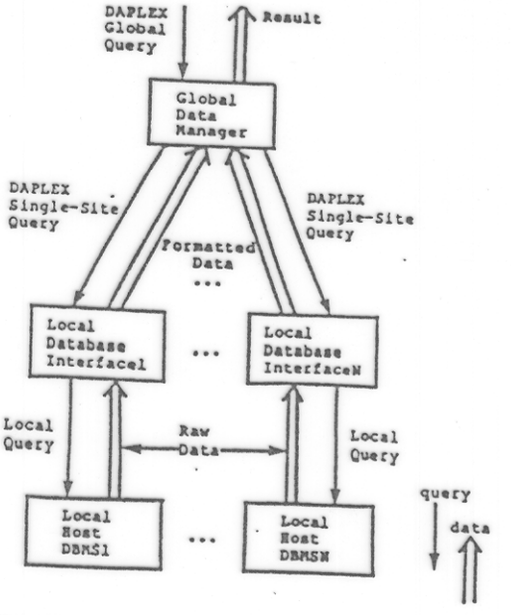
[GRS91]

Georgakopolous

Multibase

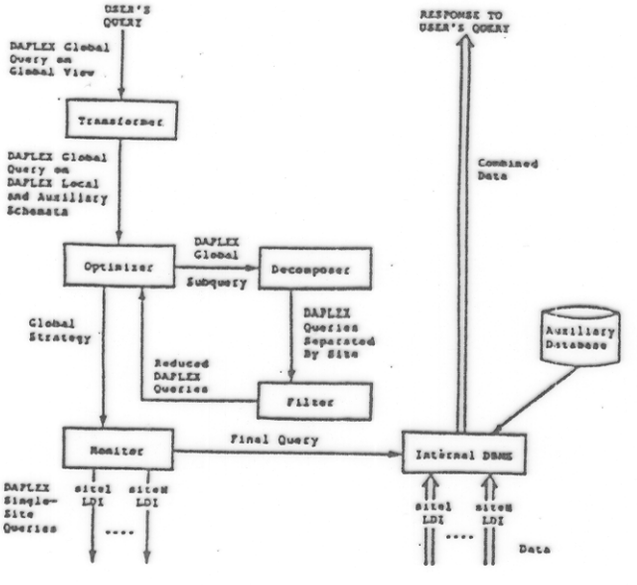


Multibase System Schematic

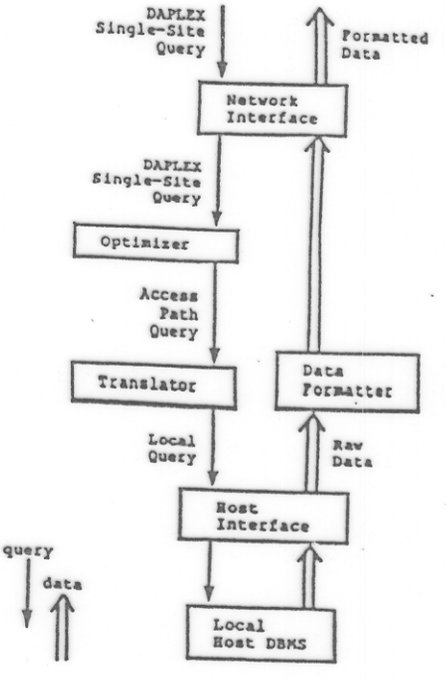


Multibase System Architecture

Multibase

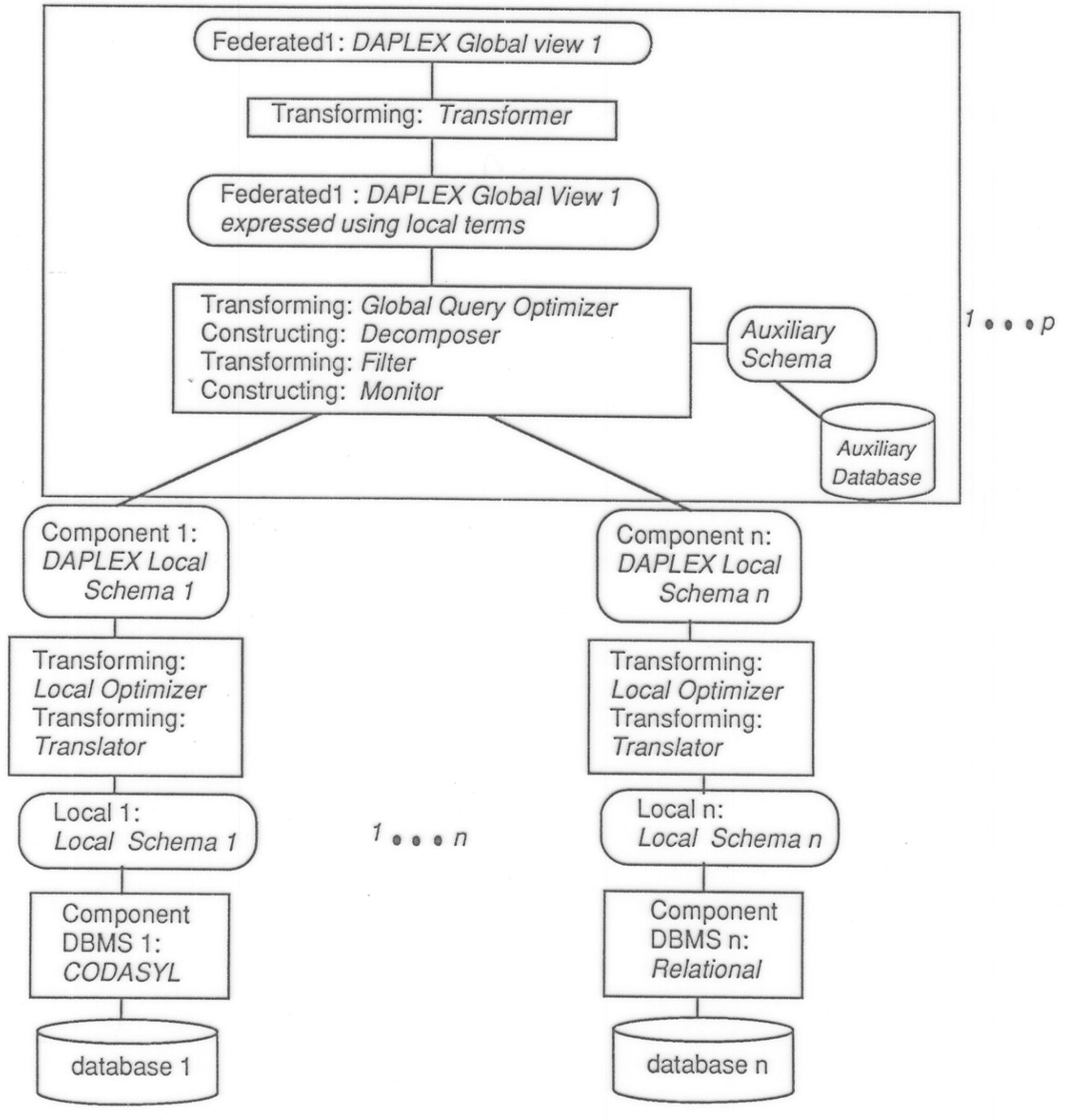


Global Data Manager Architecture



Local Database Interface Architecture

Multibase Architecture



[Landers and Rosenberg 82]

[Sheth and Larson 90]

DDTS
Important Features

Started in 1979 at Honeywell.

Schema Architecture:

Five Level Schema Architecture
(see section 2.4, page 31)

System Architecture:

Application Processor
Data Processor
(see pages 28 and 29)

Communication: DSA
(MAP in a second similar system)

Common Data Model: Relational

Conceptual Data Model: E-C-R

[Dwyer and Larson 87

Geographically Distributed (Minneapolis, Phoenix)

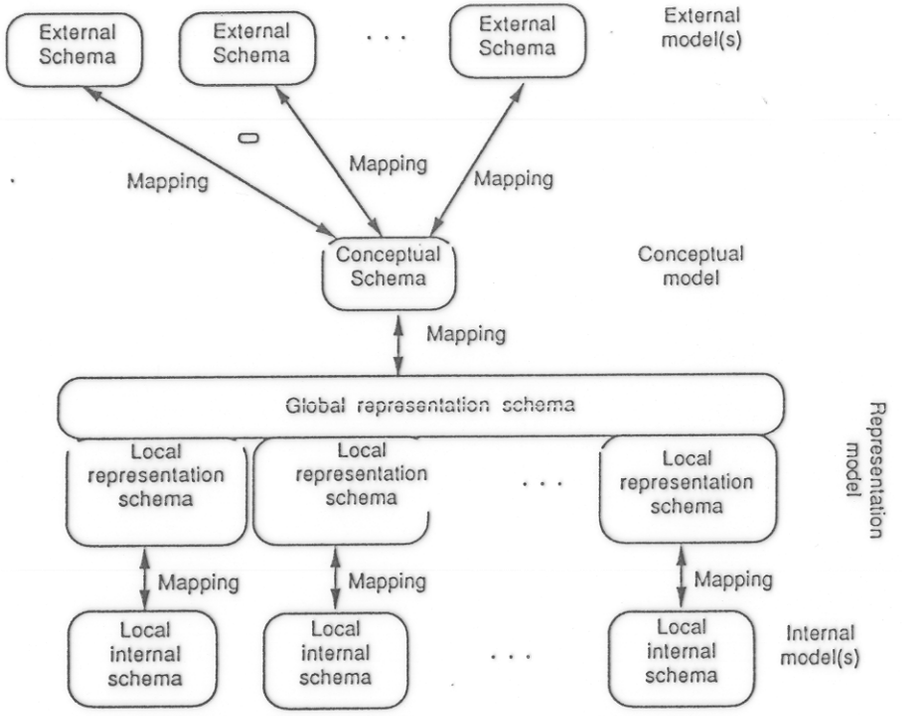
Heterogeneous Hardware, Operating System, DBMSs.
Multiple user interfaces-
GORDAS Language, SQL, and Graphical.

Current System Configuration:
Four Nodes:
Three Honeywell DPS-6 with IDS-II DBMS
One DPS-8 with RAM DBMS
Also ties to another HDDBMS called
Shop Floor Two Level Query System
(Three Node, DDTS like system with
different Hardware, OS, DBMSs)

Compiled Queries can be stored.
Allowed updates in past. Changed Operating System for Concurrency Control.

Lacks good performance because of prototype nature.

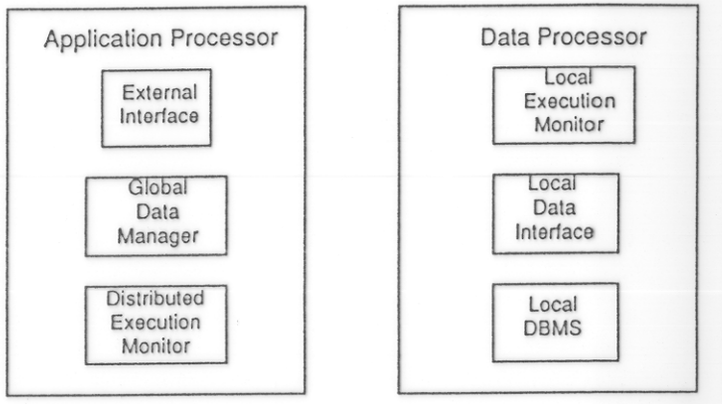
System Archetecture of DDTS: Schemas in DDTS Terminology



Adding local and global representation schemas allow better representation of the data distribution and DBMS heterogeneity, as well as provide data independence and multiple views. Many systems use same model at representation and conceptual level, thus degenerating to a four level schema architecture.

[Devor et al 82]

System Archetecture of DDTS: Processors in DDTS Terminology



External Interface (EI) translates a set of commands (transaction) on the conceptual schema in to a transaction on the cannonical (representation) schema. It may also enforce semantic integrity constraints.

Global Data Manager (GDM) translates a transaction into subtransactions for different sites at which required data reside . This function is also called global access planning. It may perform global optimization. GDM is also called Materialization and Access Planner.

Distributed Execution Monitor (DEM) coordinates exections of subtransactions at different sites.

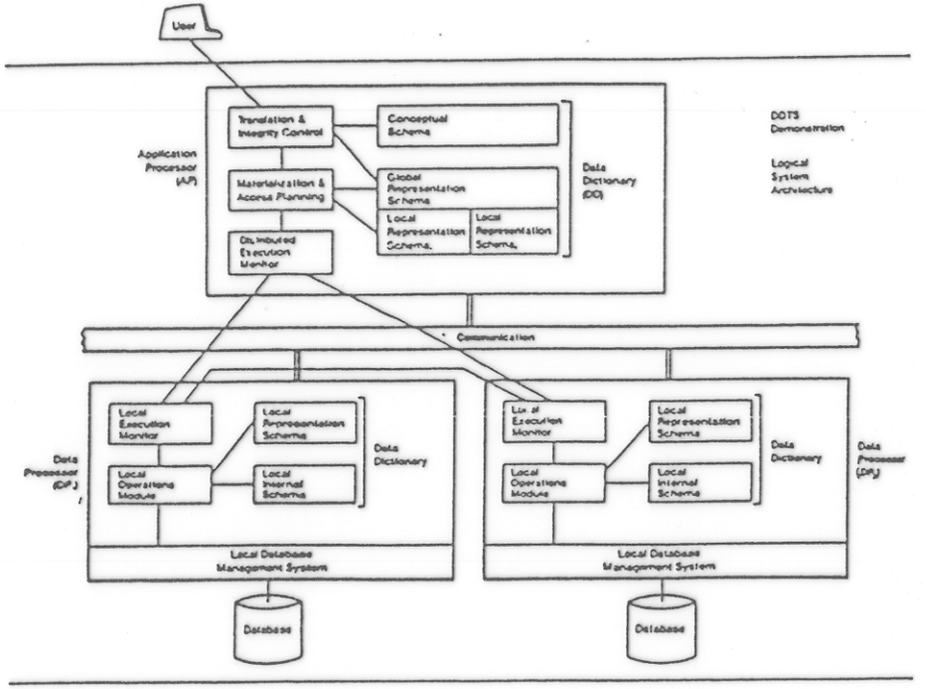
Local Execution Monitors (LEM)manages subtransaction evaluation at the local site and interacts with DEM.

Local Data Interface(LDI) translates the subtransaction in cannonical model into a set of commands in data model of local DBMS.

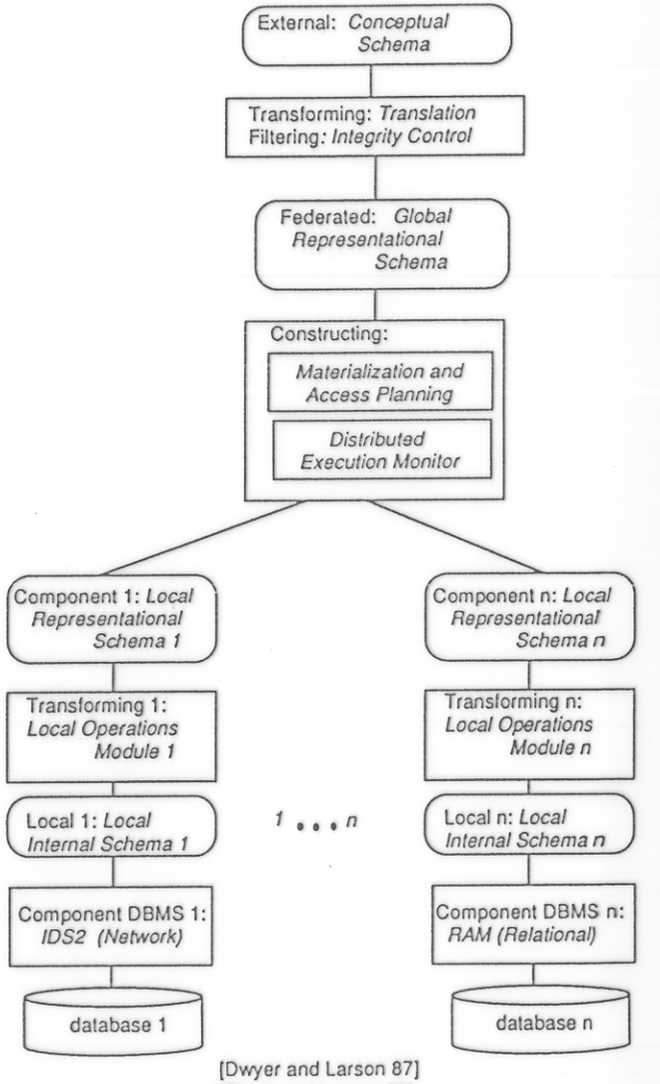
[Devor et al 82]

DDTS

System and Schema Architecture



DDTS Architecture



[Dwyer and Larson 87]
[Sheth and Larson 90]

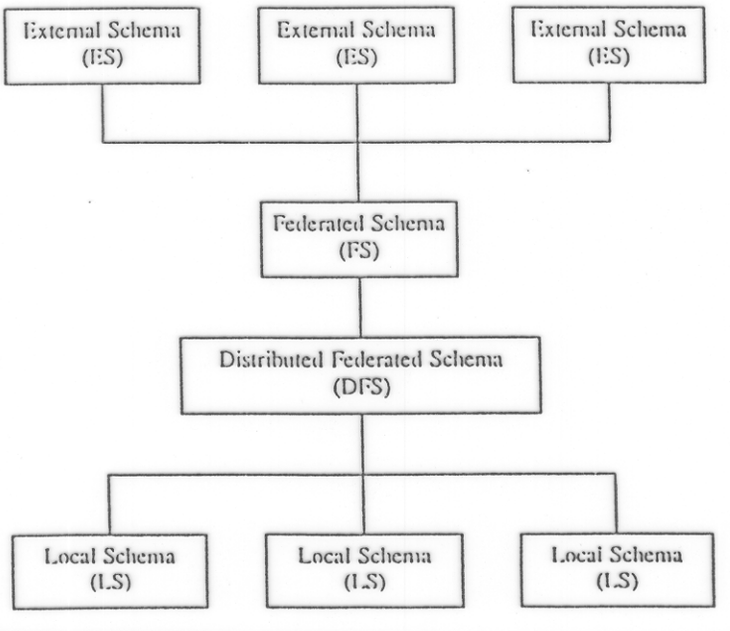
MERMAID: Important Features

- Provides the illusion of a single DBMS
- Accesses existing databases through the existing DBMS
- Provides an integrated view of the multiple underlying databases
- A user may express a query in a standard language using the relations in the integrated view
  - — MERMAID determines how to answer the query
  - — MERMAID sends subqueries to each DBMS site
  - — Translators translate the subquery to the DBMS query language
  - — The answer is integrated into a common report
- Support for multiple federated schemas
- Extensive query optimization

M. Templeton

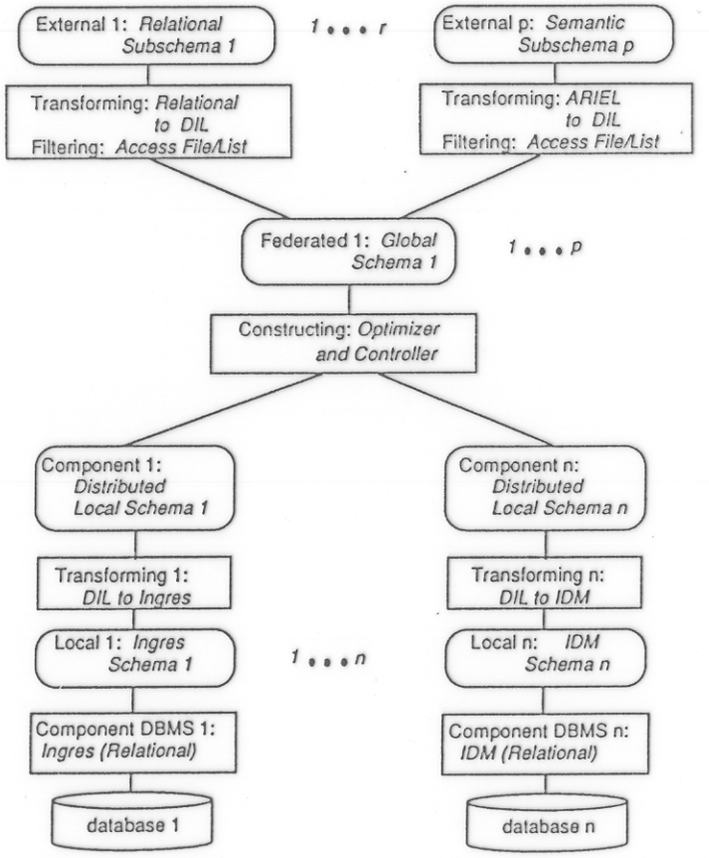
CASE STUDIES
MERMAID

SCHEMA ARCHITECTURE



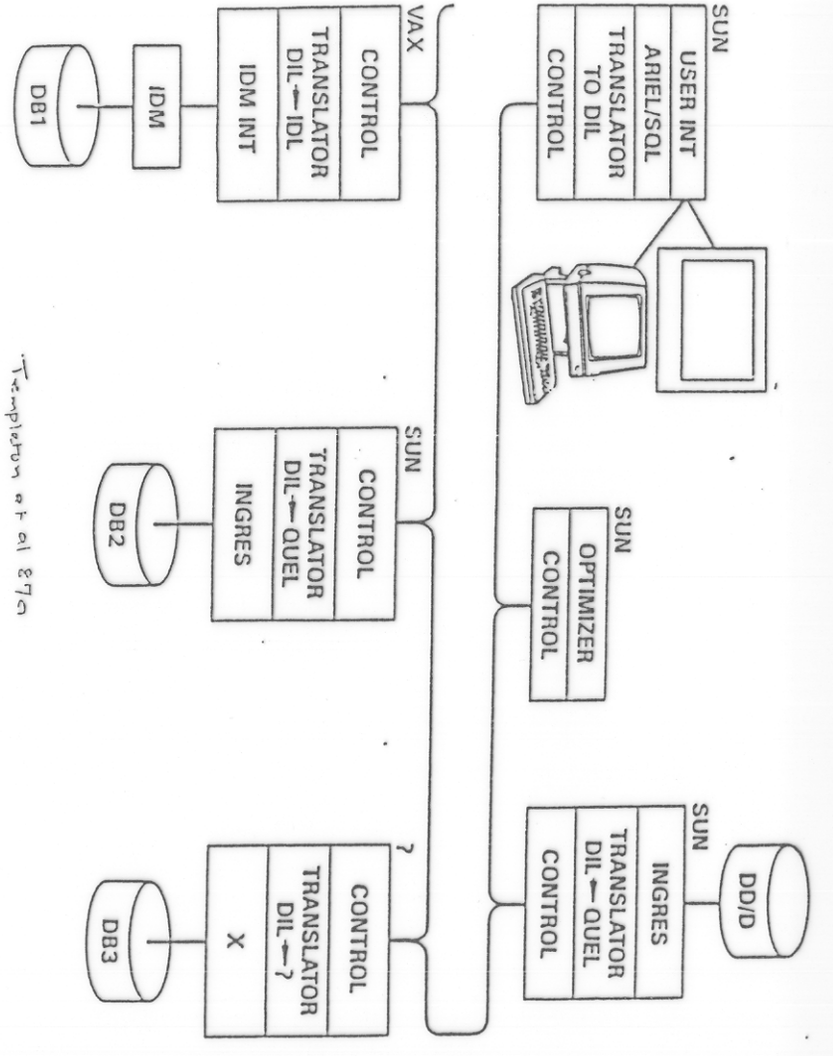
Templeton et al 87a

Mermaid Architecture



[Templeton et al 87]
[Sheth and Larson 90]

MERMAID: SYSTEM ARCHITECTURE



Templeton et al 87

MRDSM
Important Features

Datamodel homogeneous, but semantically heterogeneous

No global schema but (a) system provides functions for manipulating data belonging to distinct/nonintegrated schemas, and (b) users face a variety of views of reality

Functionality:

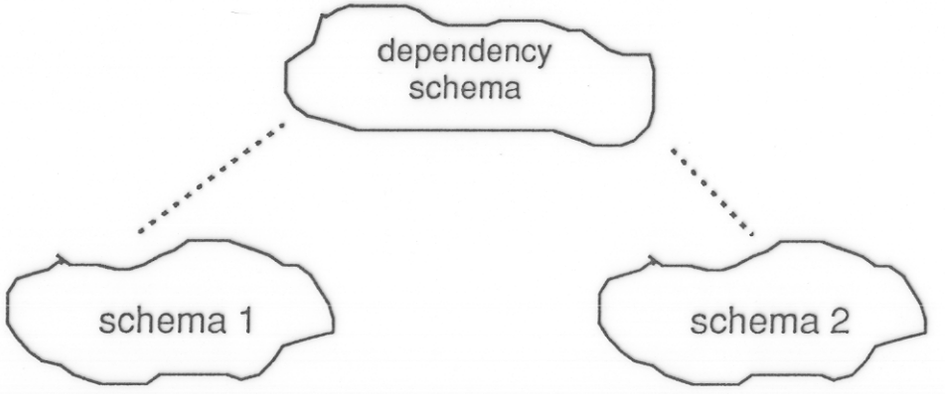
A. DBA can define dependency schema which describe interdatabase relationships in terms of privacy, integrity, data meaning, etc.

B. Data sublanguage to perform:

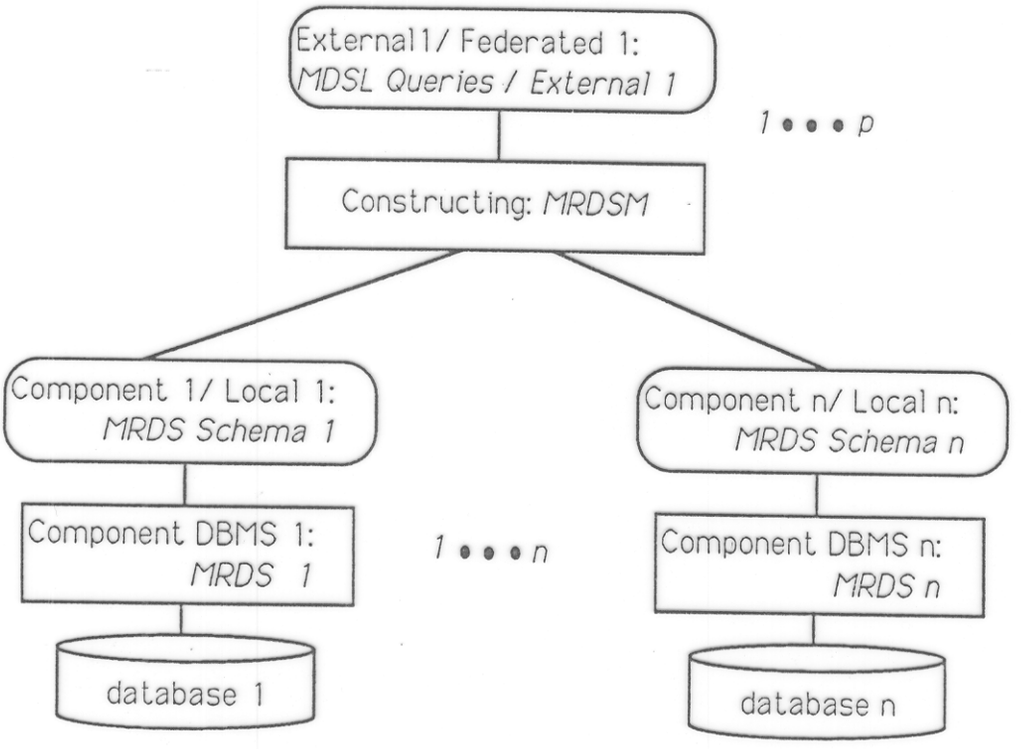
- joining of data in different database schemas
- user defined value types for dynamic trformation of attributes
- interdatabase queries
- dynamic aggregration of data, etc.

Interesting features:

- multiple identifiers
- semantic variables
- dynamic attributes
- interdatabase queries



[Litwin and Abdellatif 86]



MRDSM Architecture

[Sheth & Larson 90]

Appendix: Features of Some FDBS/Multi-DBMS Efforts

Table A.1 summarizes the choice of a common data model and the types of component DBMSs integrated into some of the experimental prototype systems. Table A.2 gives an empirical and partial list of significant or interesting features of some of these efforts. No effort is made to present all systems that exhibit FDBS features or to capture all important details of those mentioned.

Table A.1. Data Models of Various Multi-DBMS/FDBS Efforts

Effort	CDM	Types of Component DBMSs	Reference
ADDS	Extended Relational	Hierarchical, Relational, Files	[Breitbart et al. 1986]
CALIDA	Relational like	Relational, Relational like, Files	[Jacobson et al. 1988]
DDTS	Relational*	Network, Relational	[Dwyer and Larson 1987]
DQS	Relational	Hierarchical, Network, Relational	[Belcastro et al. 1988]
IISS	E-R Variant (IDEF1x)	Network, Relational	[IISS 1986]
Mermaid	Relational (DIL)	Relational (partial integration of text)	[Templeton et al. 1987b]
MRDSM	Relational	Relational	[Litwin 1985]
Multibase	Functional (DAPLEX)	Network, Relational	[Landers and Rosenberg 1982]
OMNIBASE	Extended Relational	Relational	[Rusinkiewicz et al. 1989]
SIRUS-DELTA	Relational	Network, Relational	[Litwin et al. 1982]
SCOOP	E-R	Hierarchical, Network, Relational	[Spaccapietra et al. 1982]

* Extended E-R is used for integrity constraint enforcement at the federated schema level and at the external schema level. The primary CDM can be said to be relational since the internal command language is based on the relational algebra.

Table A.2. Significant Features

Effort	Significant Features
ADDS	Tightly coupled, limited updates, in-house use of the prototype system
CALIDA	More like loosely coupled, intermediate language, interactive, menu-driven user interface, data dictionary editor/browser, in-house use of the prototype system
DDTS	Tightly coupled architecture, integrity constraint checking, local query optimization, use of local and long haul communication
DQS	Tightly coupled, completeness of implementation, auxiliary schema, IBM environment, attention to autonomy
IISS	More like tightly coupled, queries must be compiled, forms-based user interface
Mermaid	Tightly coupled, completeness of implementation, data dictionary, access control, global query optimization, extensions to include texts, experience with communication systems, workstation environment
MRDSM	Loosely coupled architecture, multidatabase language, dealing with multiple semantic interpretations
Multibase	Tightly coupled, completeness of implementation, schema integration, global query optimization, auxiliary schema, architecture, number of component DBMSs integrated in various prototypes
OMNIBASE	Loosely coupled, DEC environment, query processing, distributed DBMS as a component DBMS

Sheth & Larson 90

What is semantics?

"the scientific study of the relations between signs and symbols and what they denote or mean".*

So we have *sign and symbols* => model, language

- this is structured, limited, incomplete

what they *mean or denote* => real world information

- some times called real world semantics)
- this is unstructured, unlimited, possibly complete

One way to make a model useful is to talk about modeling in a specific *domain of discourse* [i.e., context]- hence limiting amount of information modeled (semantics captured).

*Reference: William Woods, "What is a Link?."

A critical issue in Information Modeling:

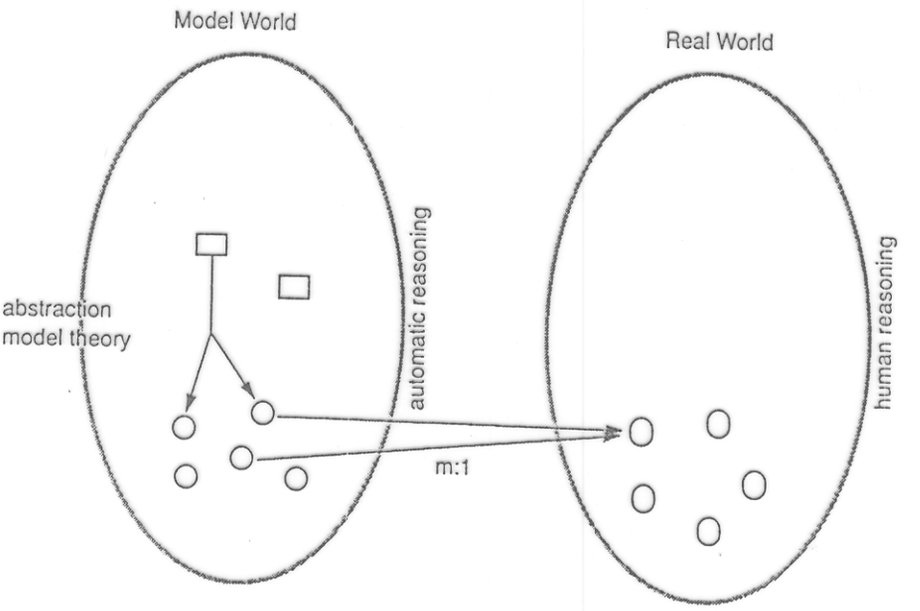
- Identify semantically related object
- Determine schematic differences

Modeling Semantics:

- Context
- Abstractions
- Domains
- States

[Sheth and Kashyap 92, papers in Sheth 91b, Kent 91]

Automatic vs. Human Reasoning



Schema S1

Schema S2