PHENIX project

BIKIT-FUNDP

IRSIA/IWONL "Tronc Commun" - nb. 5421

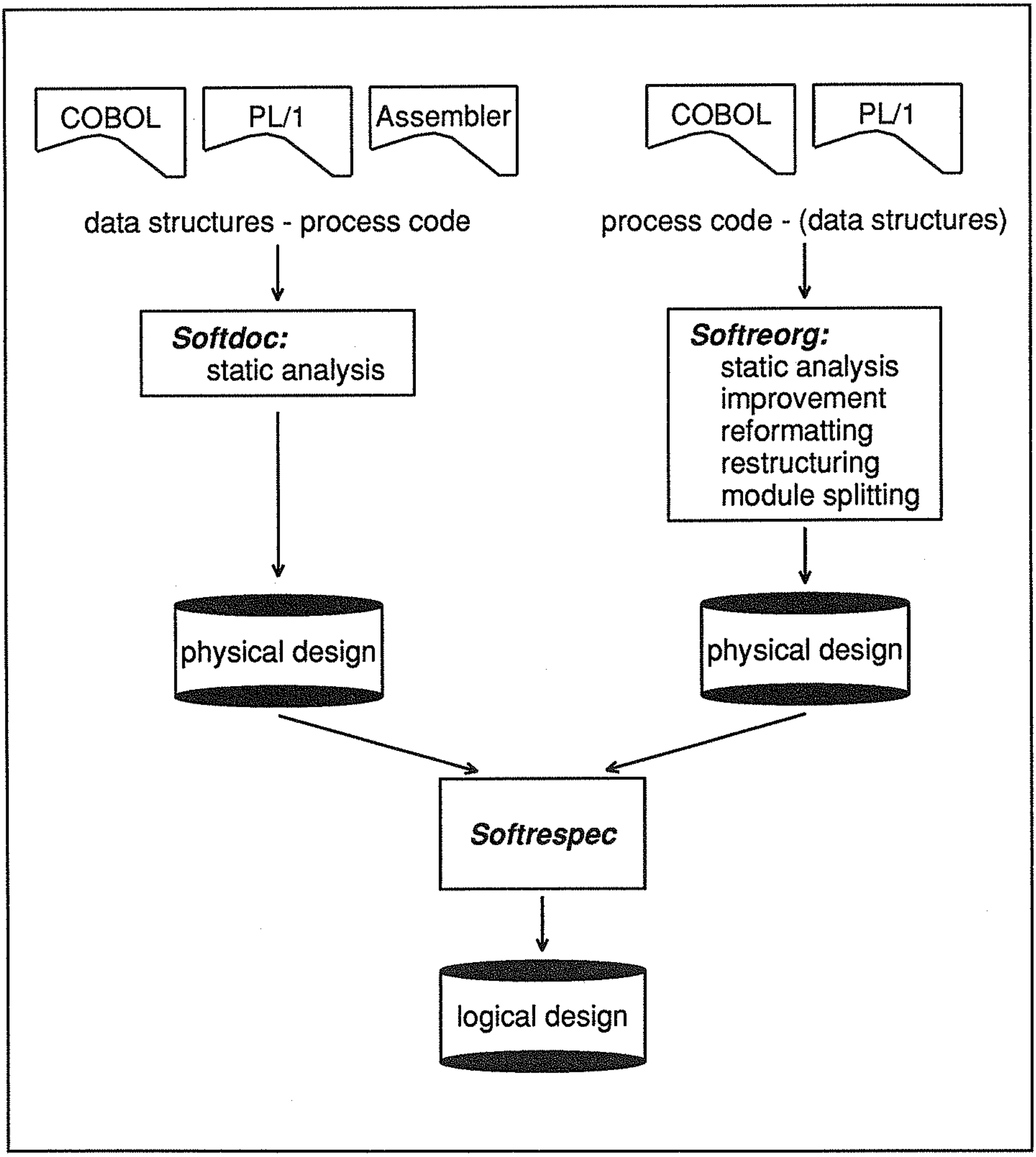
Phenix Symposium on Reverse Engineering of Databases

Namur

June 1st-2nd, 1992

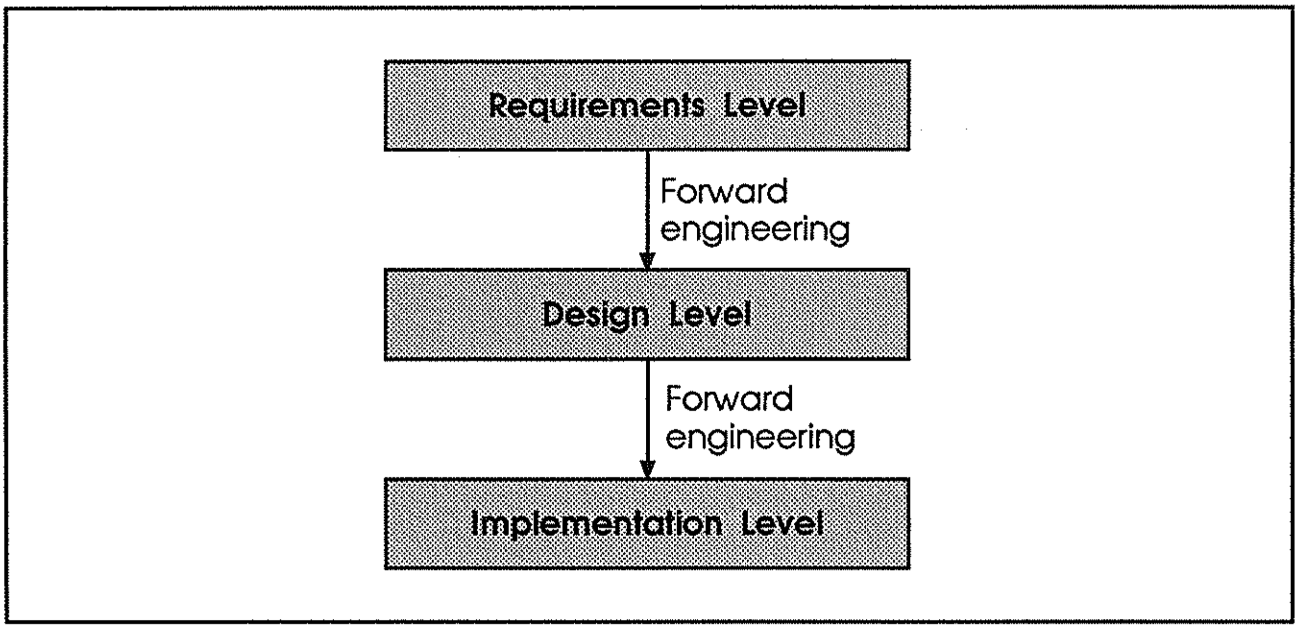
Introduction to reverse engineering

Softorg *RE* environment



1. The software maintenance Crisis

1.1. Levels of Abstraction in Software Systems.



- Requirements level : specification of the user requirements.
- Design level : description of an implementation-independant solution.
- Implementation level : description of a solution using current technology.

1.2. Evolution of software systems (Maintenance).

Laws of Lehman.

- Systems are dynamical in nature and are due to change.
- As systems change they become more and more complex if no remedial action is taken.

=> Changes to a system must be propagated through the different levels of abstraction.

-- Changes in user requirements.

-- New technology becomes available.

=> Under these conditions :

-- Documentation on the system will be complete, correct and up-to-date.

-- Subsequent changes are easier to conduct.

1.3. Current state of software systems.

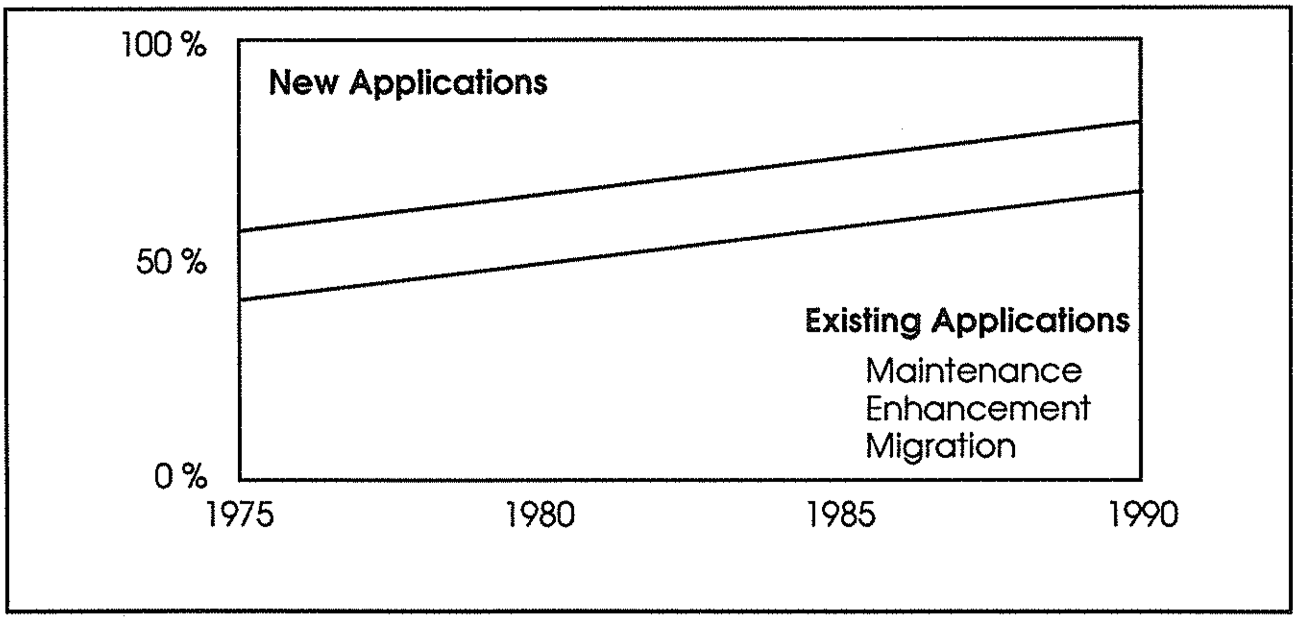
- For old systems : no documentation available.
- For more recent systems : documentation is not complete, up-to-date or correct.

=> Only reliable description of the system is the source code of the system.

=> Much time is spent trying to understand existing system.

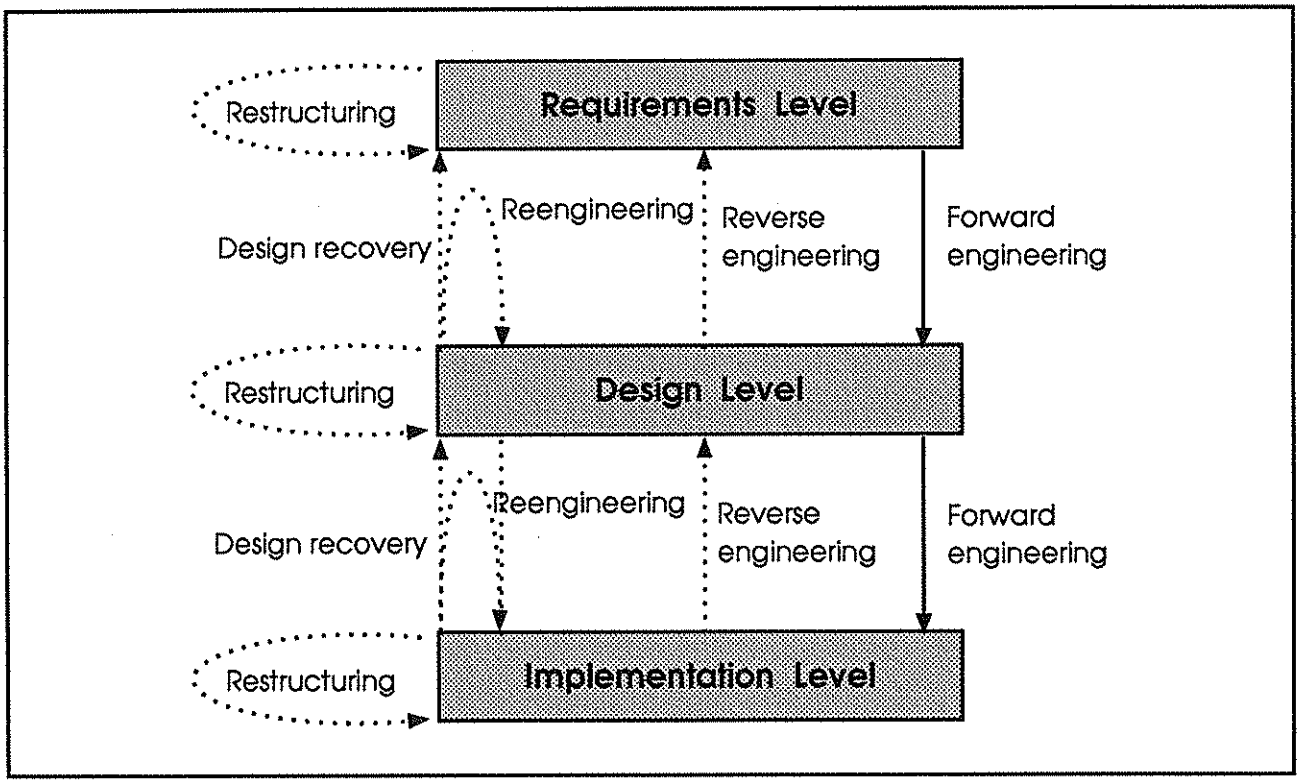
=> Some changes are made ad hoc on the source code level.

=> System becomes even more difficult to be maintained.



2. Reverse engineering & related Concepts.

2.1.Definitions of terms.



Forward engineering :

traditional process of moving from high-level abstractions and logical designs to the physical implementation of the system

Reverse engineering :

Process of analyzing the subject system to identify the system's components and their interrelationships and to create representations of the system in another form or at higher levels of abstraction.

=> Functionality of the system does not change !

=> Generic concept :

Redocumentation :

Creation or revision of a semantical equivallent representation within the same abstraction level.

ex : creation of call-graph from source code.

Design Recovery :

Subset of reverse engineering in which domain knowledge, external information, and deduction or fuzzy reasoning are added to the observations of the subject system to identify higher level abstractions.

=> Knowledge based approach : Phenix system.

Restructuring :

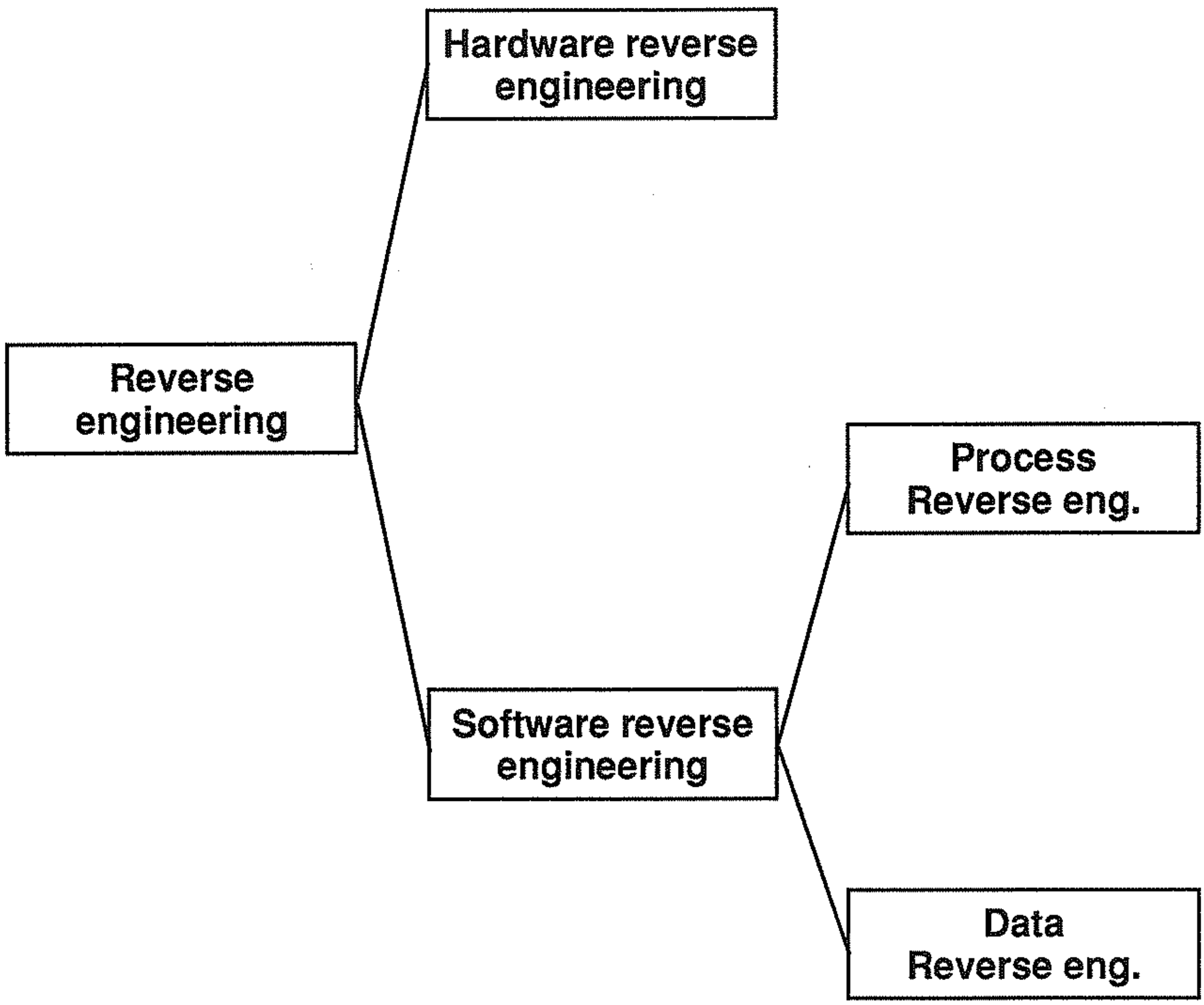
Transformation from one representation to another at the same abstraction level, while preserving the systems external behaviour.

ex : code-to-code transformations, normalization

Reengineering :

Examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form.

2.2.Application domains & History.



3. Objectives of reverse engineering

3.1. Application reengineering.

Optimalization of existing applications to satisfy new conditions.

3.2. Application maintenance.

Slight modifications of applications to correct bugs or reflect evolutionary change.

3.3. Application redocumentation.

Get accurate and correct documentation on the system.

3.4. Application conversion.

Migration of the system due to environmental changes.

ex : IMS --> Relational systems.

3.5. Application integration.

Merging of independent applications into a single one.

ex : integration of underlying databases

3.6. Application development.

Reverse engineering the current application will help development of new ones.

State-of-the-art in Data Base Reverse Engineering

Contents

1. Reverse Engineering of Process Code
2. Data Base Reverse Engineering: Research Projects
3. Data Base Reverse Engineering: Commercial Tools
4. *Phenix* Project
5. Future Trends

Reverse Engineering of Process Code

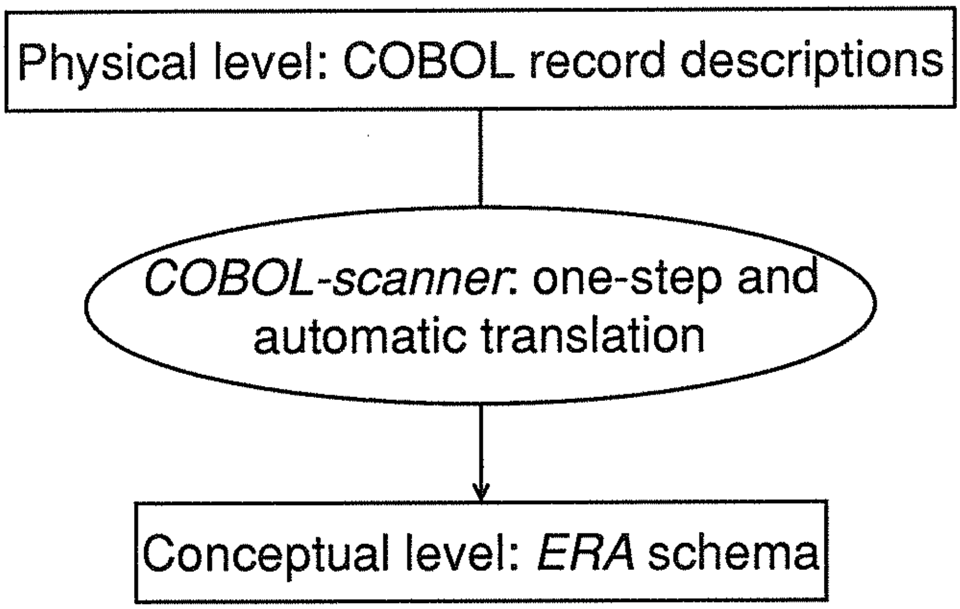
Types of tools:

1. Static and dynamic analysis tools
2. Documentation tools
3. Code converters
4. Code improvement tools:
   a. reformatters
   b. restructurers
   c. module splitters
   d. debuggers
   e. data standardisers
5. *True* reverse engineering tools
   a. Definition: abstraction of program code to a logical or conceptual design and store this in a repository
   b. Availability: very few
   c. Products: Domino, Aisle, and Synon/2

Research projects: *REDO and PRACTITIONER Esprit* projects

Data Base Reverse Engineering: Research Projects

Nilsson (1985)

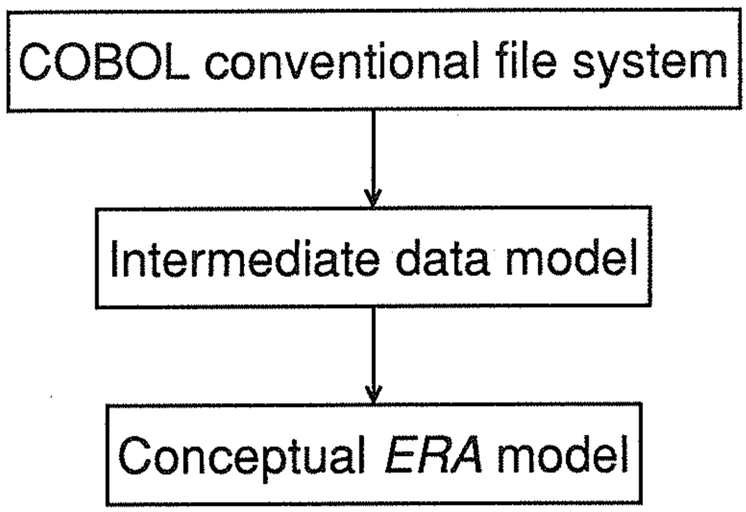


COBOL *RE* related problems:

- transformation of names
- transformation of tables
- redefinitions

Davis & Arora (1985)

Methodology



COBOL *RE* related problems:

- access keys
- constraints

Knowledge-based implementation of methodology

REDO Esprit Project

OBJECTIVES:

- development of methodology and tools for *software maintenance*
- coherent methodological framework for *reverse engineering*

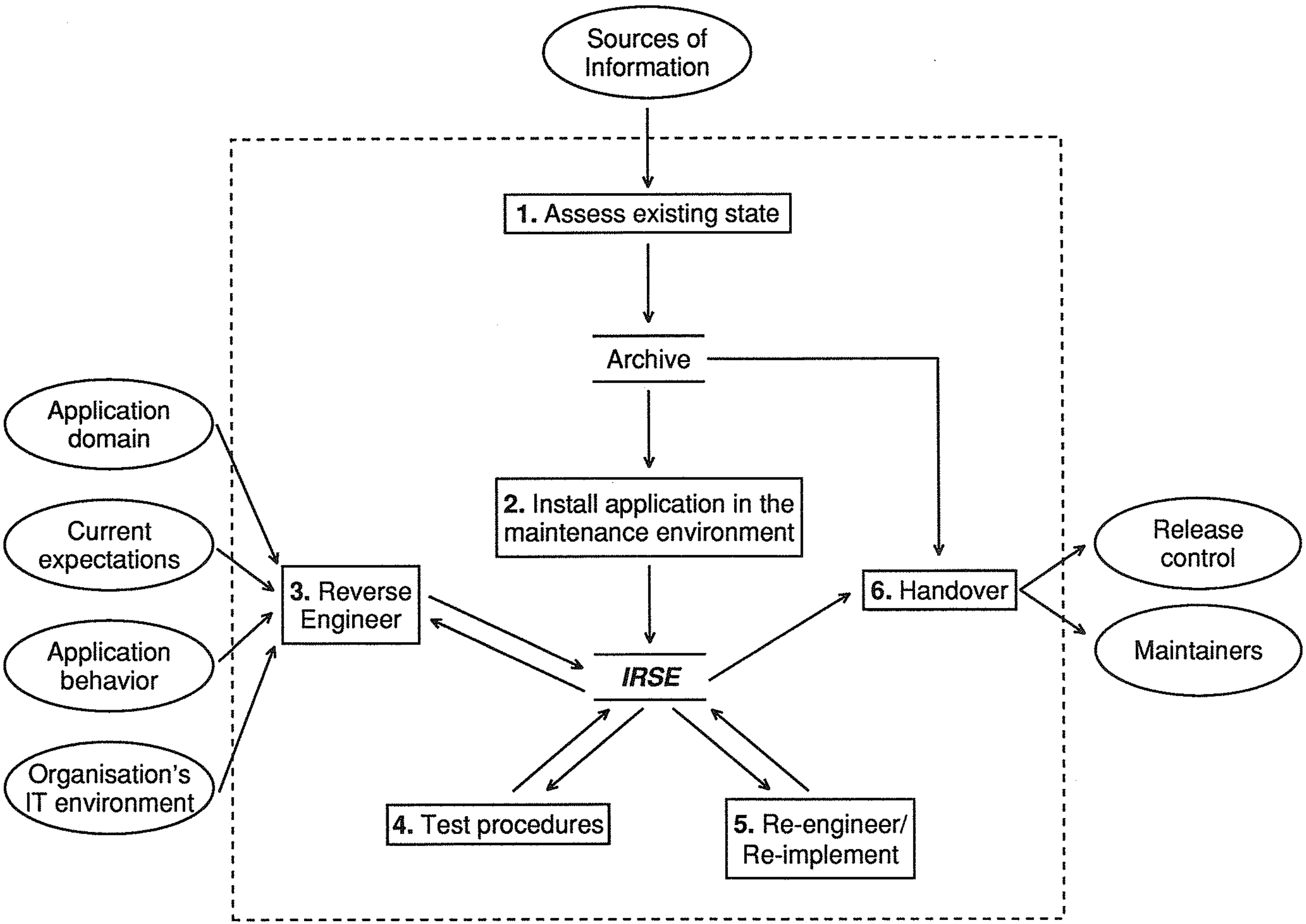
APPLICATION DOMAINS:

- real time systems
- *DBMS*
- user interface
- high performance computers (vector and parallel processors)

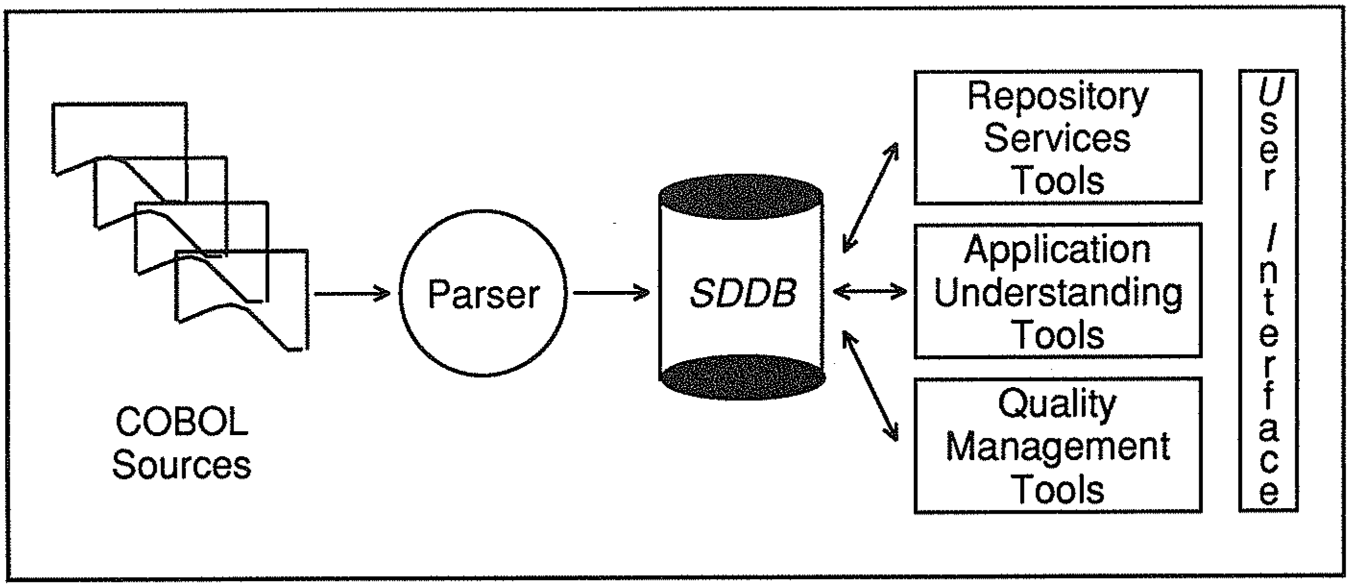
THE *REDO* METHODOLOGY:

- *generic* method for software maintenance
- refinement of the generic *REDO* method for different application domains

Generic *REDO* Method: Information Flow

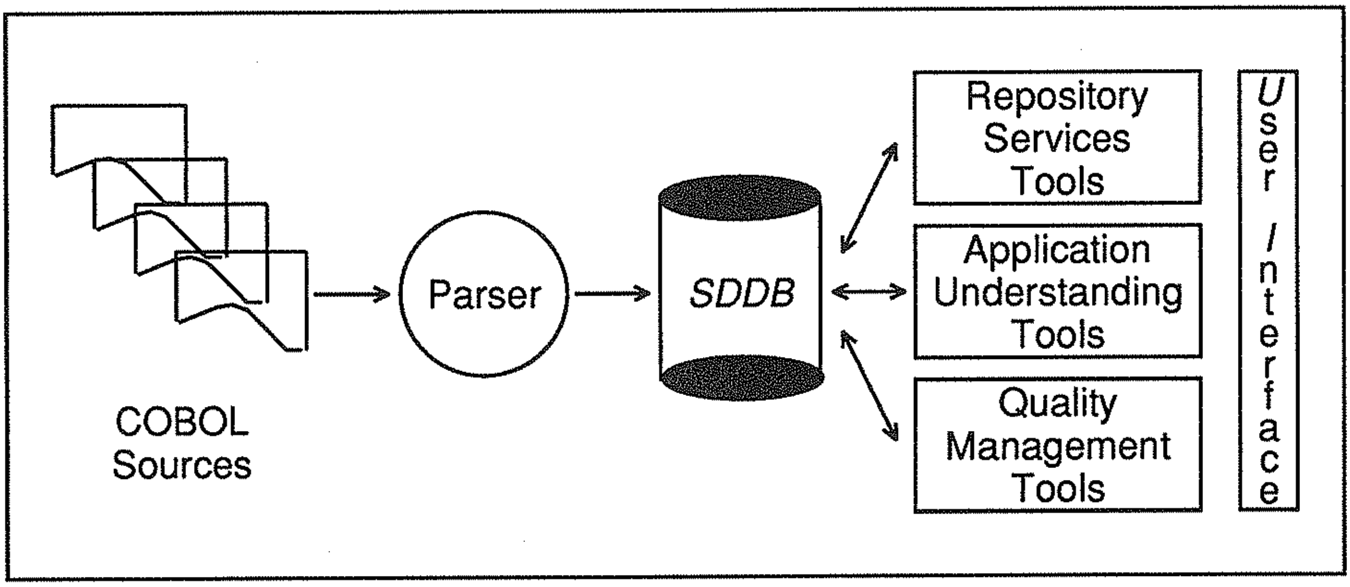


The *REDO* Architecture for Reverse Engineering



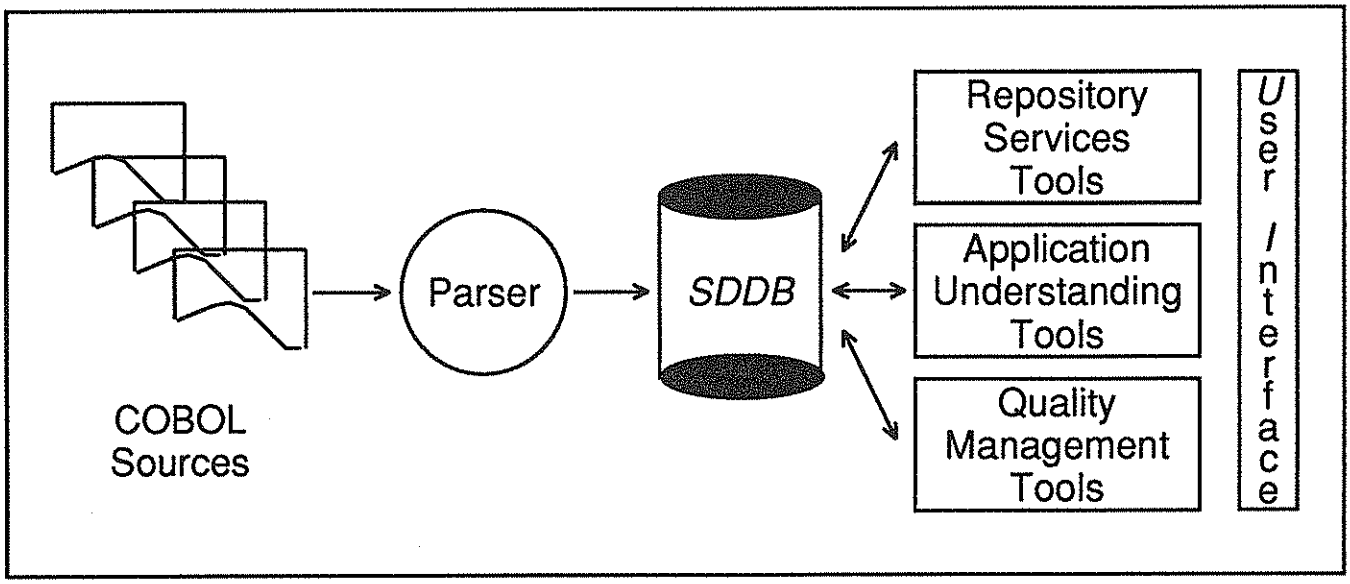
SDDB - *S*ystem *D*escription *D*ata *B*ase:

1. central repository of all information about the application
2. intermediate representation of the source code:
   a. language dependent (COBOL), *OR*
   b. language independent representation: UNIFORM



Repository Services Tools:

1. Loaders and parsers
2. Access and update facilities of the *SDDB:* editors and code viewers

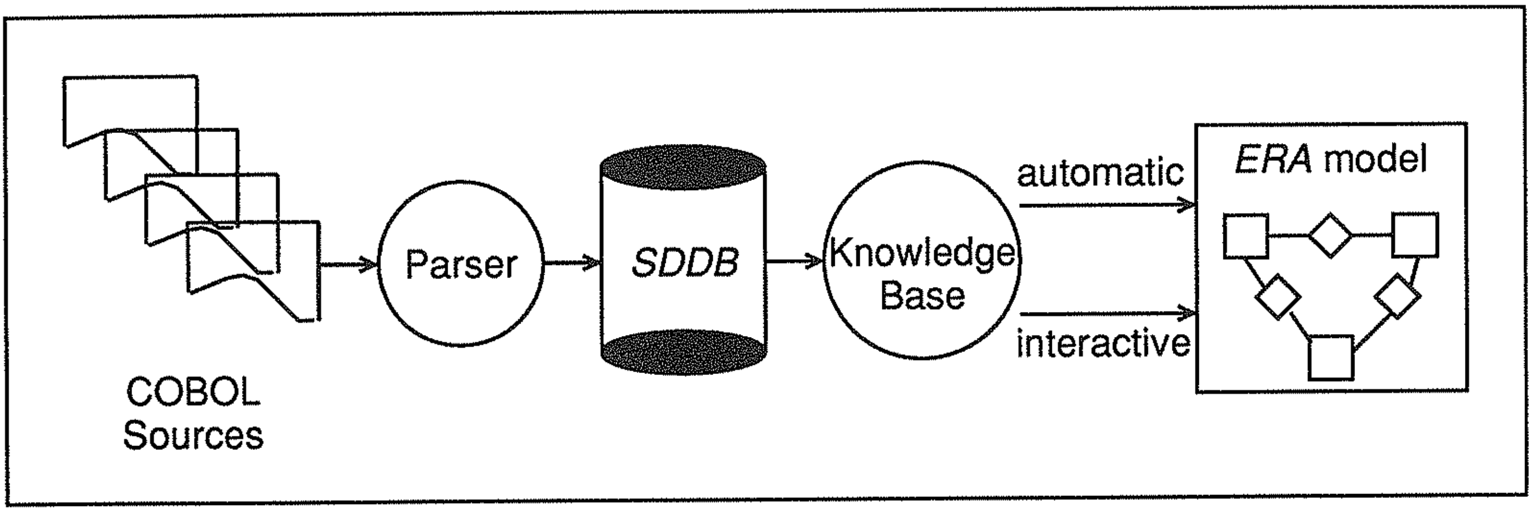


Application Understanding Tools:

1. Function abstraction
2. Restructuring
3. Module splitting
4. Data re-modeling
5. Technical documentation: Control and Data Graph Analysis
6. Documentation tools

Quality Management Tools: metrics and validation

DBRE in REDO



INPUT:

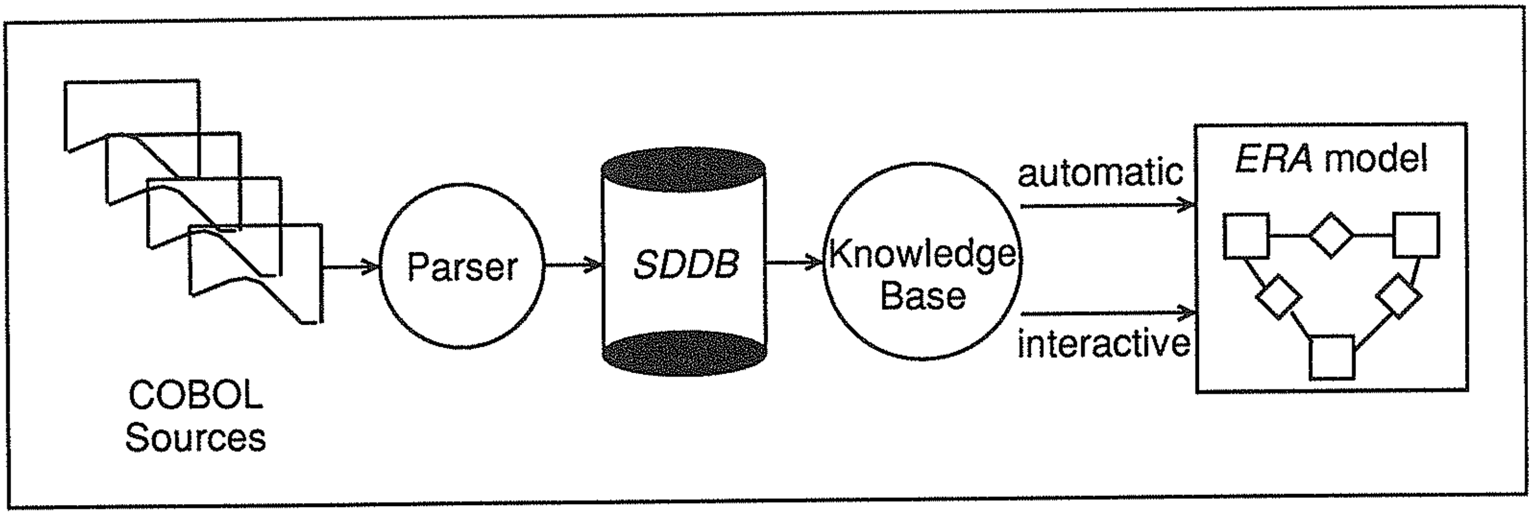
- no direct input from COBOL source, only *SDDB*
- only COBOL standard file systems
- only INPUT-OUTPUT SECTION and FILE SECTION
- no sort files and WORKING STORAGE data

OUTPUT: first cut ERA model

PROCESSING:

rules and heuristics in the knowledge base build up a *first cut ERA* model:

1. automatic mapping between COBOL data structures and *ERA* model
2. interactive: present alternative solutions to the user



Refinement and Enhancement of first cut *ERA* model:

1. interactive graphical *ERA* diagram
2. knowledge base: limited set of rules for assisting the refinement and enhancement of the *ERA* model

Maintenance of Knowledge Base:

1. *REDO* initial set of heuristics and rules
2. Reverse engineer updates knowledge base:
    a. knowledge of the application domain
    b. knowledge of the "house style", e.g., naming rules

Links between original source and derived model are maintained

Interface with other *REDO* Tools

SUMMARY

1. COBOL: largest research interest
2. COBOL presents several difficulties for *RE*
3. *DBMS* type: standard file system
4. basic framework for *DBRE*:

   DBRE = reversal of *DB* forward engineering
5. *DBRE* system contains a central repository as an intermediate (language independent) representation of source code
6. *DBRE* is an integral part of a maintenance methodology

   integration of *DBRE* tool with other tools, e.g., process code *RE* and repository
7. knowledge-based approach
8. *DBRE* tool = interactive tool
9. current research projects:
   a. extraction of information out of COBOL sources is limited
   b. limited support for conceptualization process
   c. no name processing

Data Base Reverse Engineering: Commercial Tools

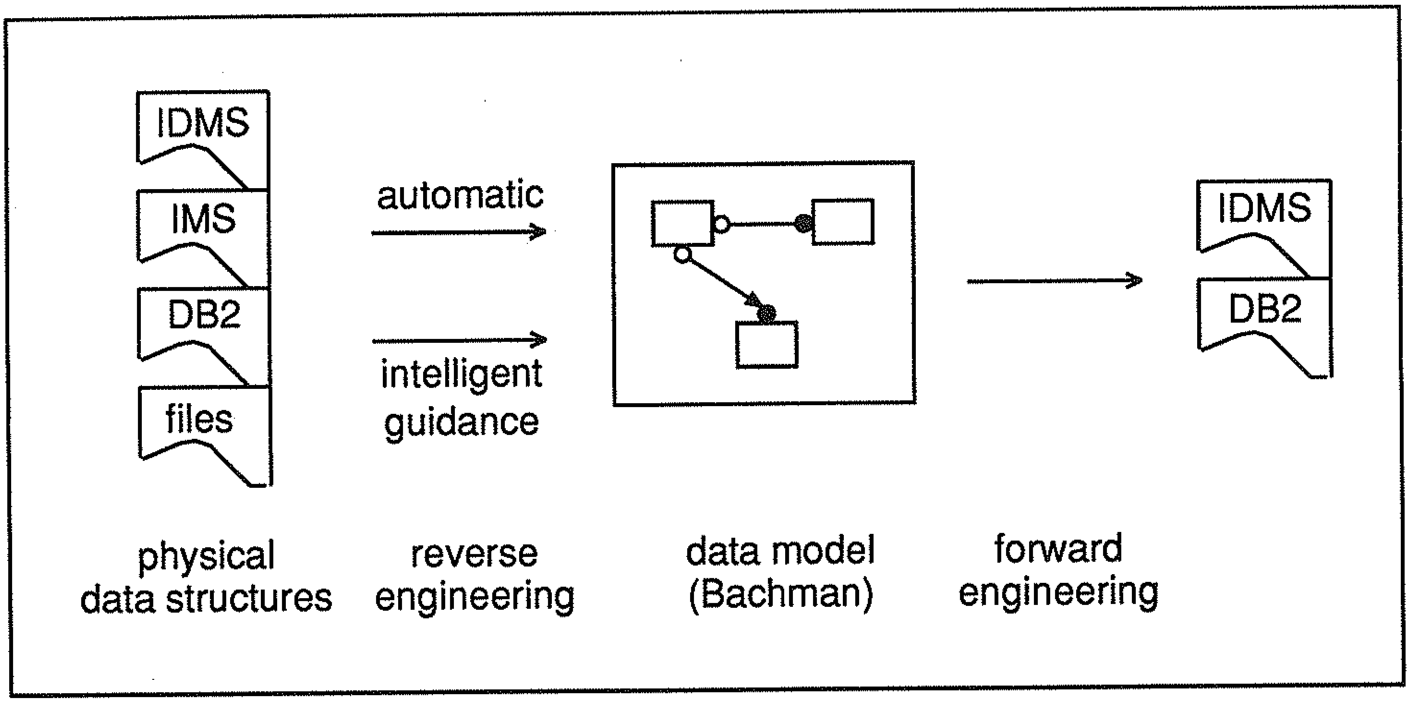
Bachman Re-engineering Toolset

Developer:	Bachman Information Systems Inc, *USA*
First issued:	1988
Target market:	IBM
Environment:	IBM PS/2 model 70 or 80 and Compaq 386 for networked PC's
Price:	£4,400-£13,600 per module
Worldwide sales:	1988 - 580 copies/product sets

Bachman Re-engineering Toolset

FE and *RE* of data

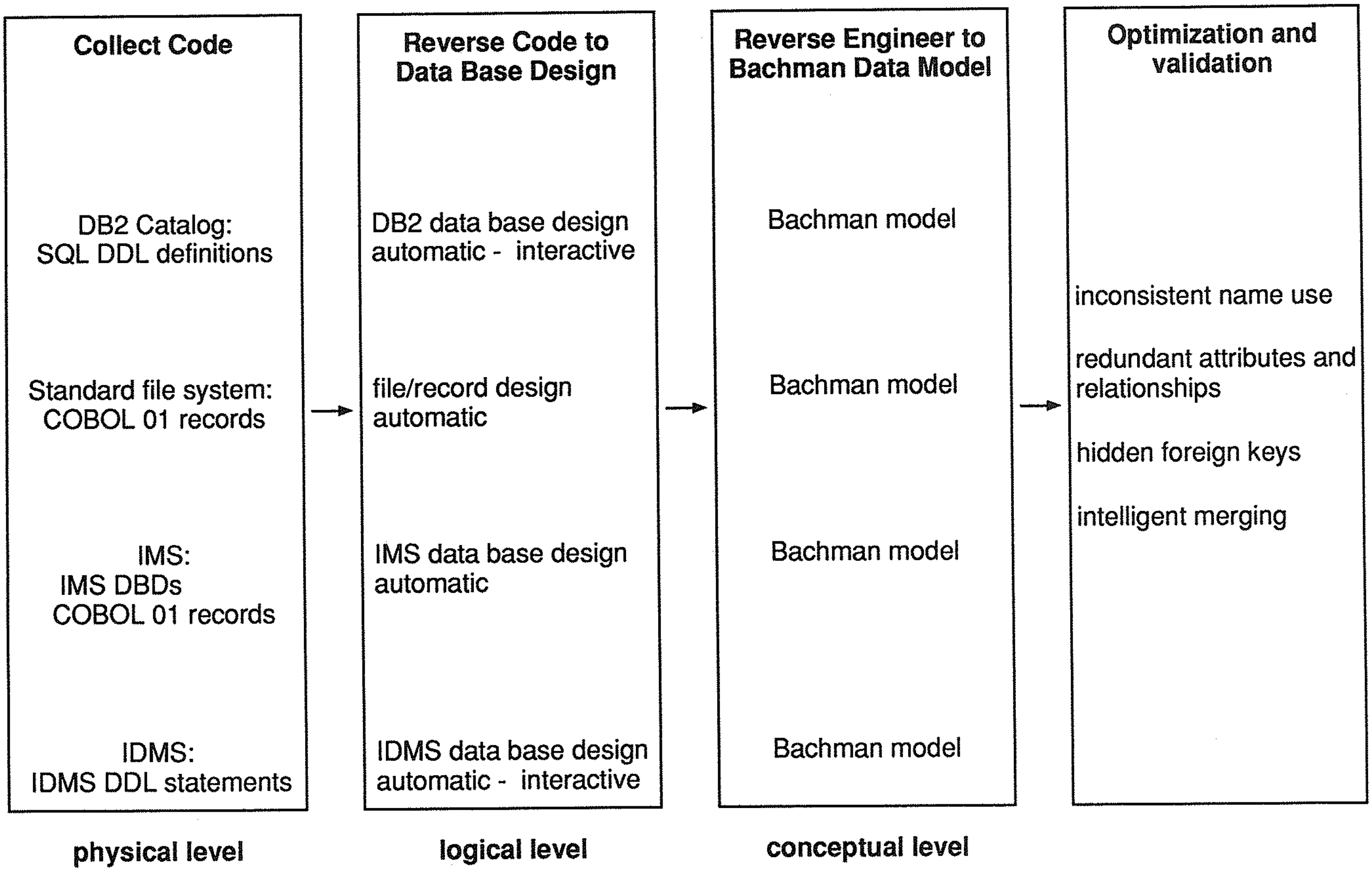
process code *RE* is under development

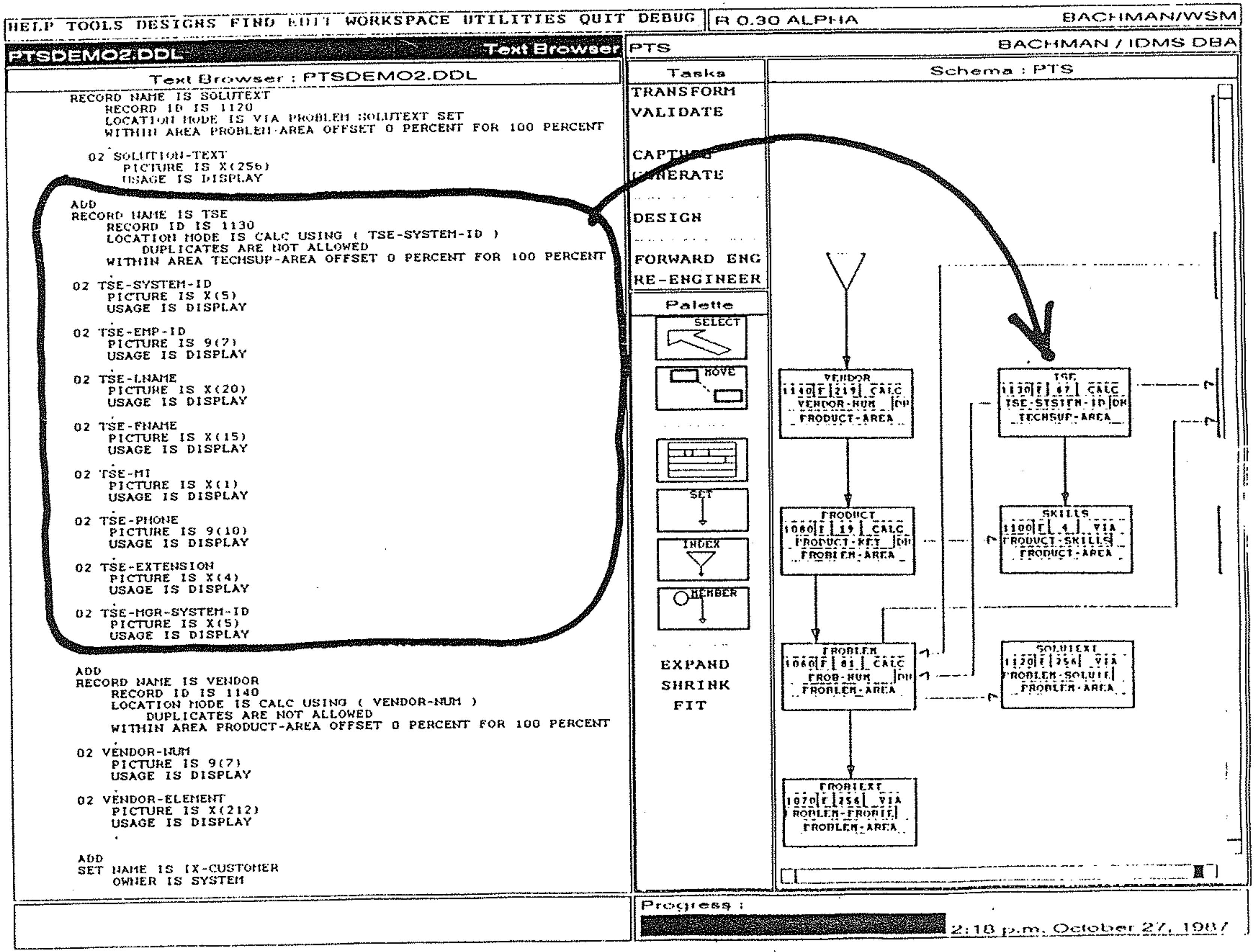


Bachman Workstation Manager:

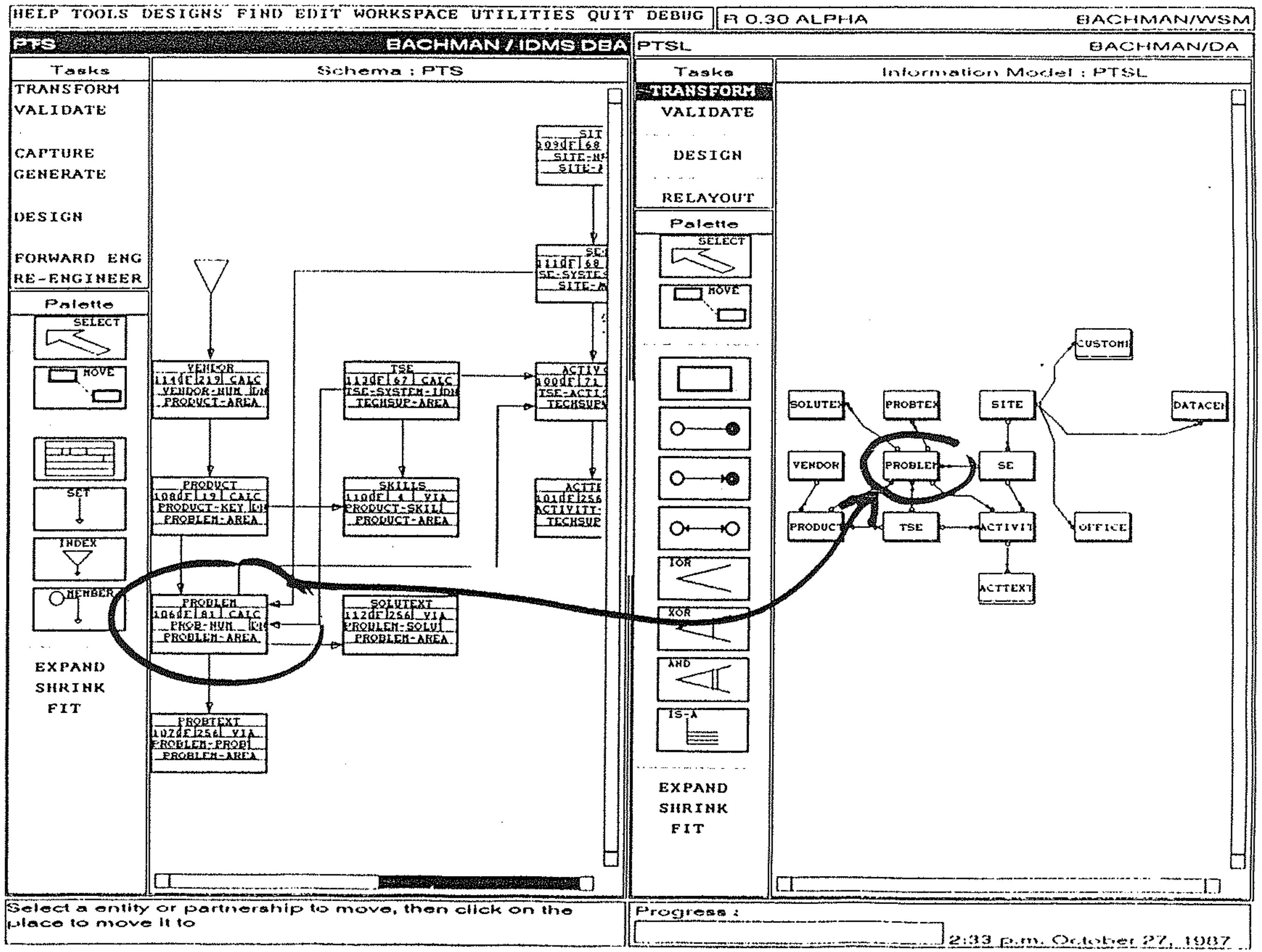
1. central repository
2. expert system rule-processing
3. *GUI*
4. session management

Bachman's Main Activities on Reverse Engineering of Data

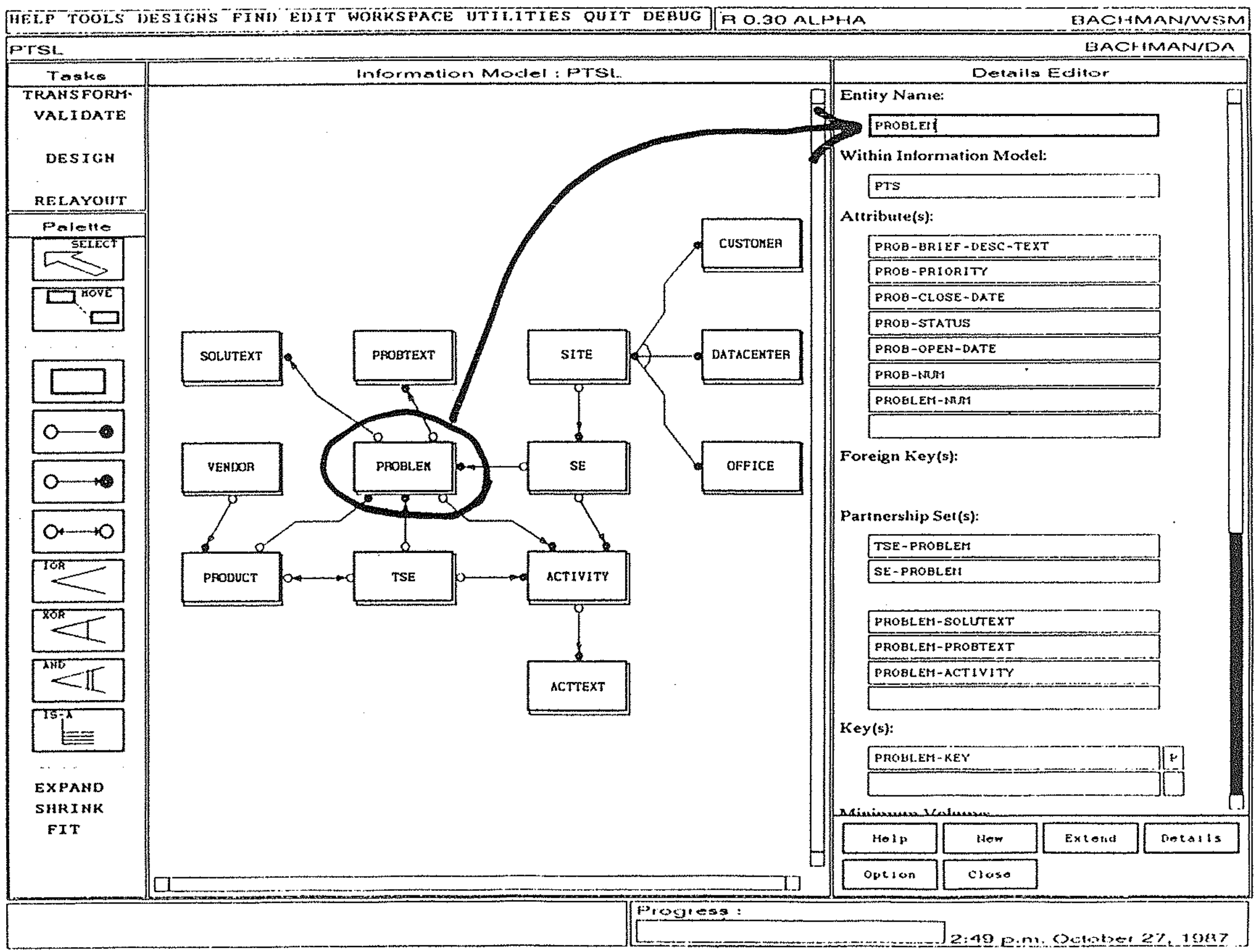




(1) IDMS data description language (DDL) statements, from the IBM mainframe, captured by BACHMAN / IDMS Database Administrator product, as Implementation level descriptions, and displayed as a "Bachman Diagram".



(2) Implementation level descriptions reverse engineered by the BACHMAN / IDMS Database Administrator product to create an Entity Diagram and its Specification level descriptions.



(3) Editing and validating Specification level descriptions by the BACHMAN / Data Analyst product to reflect new business requirements. The Detail Editor screen on the rights is examining the description of the "PROBLEM" entity.

Softorg

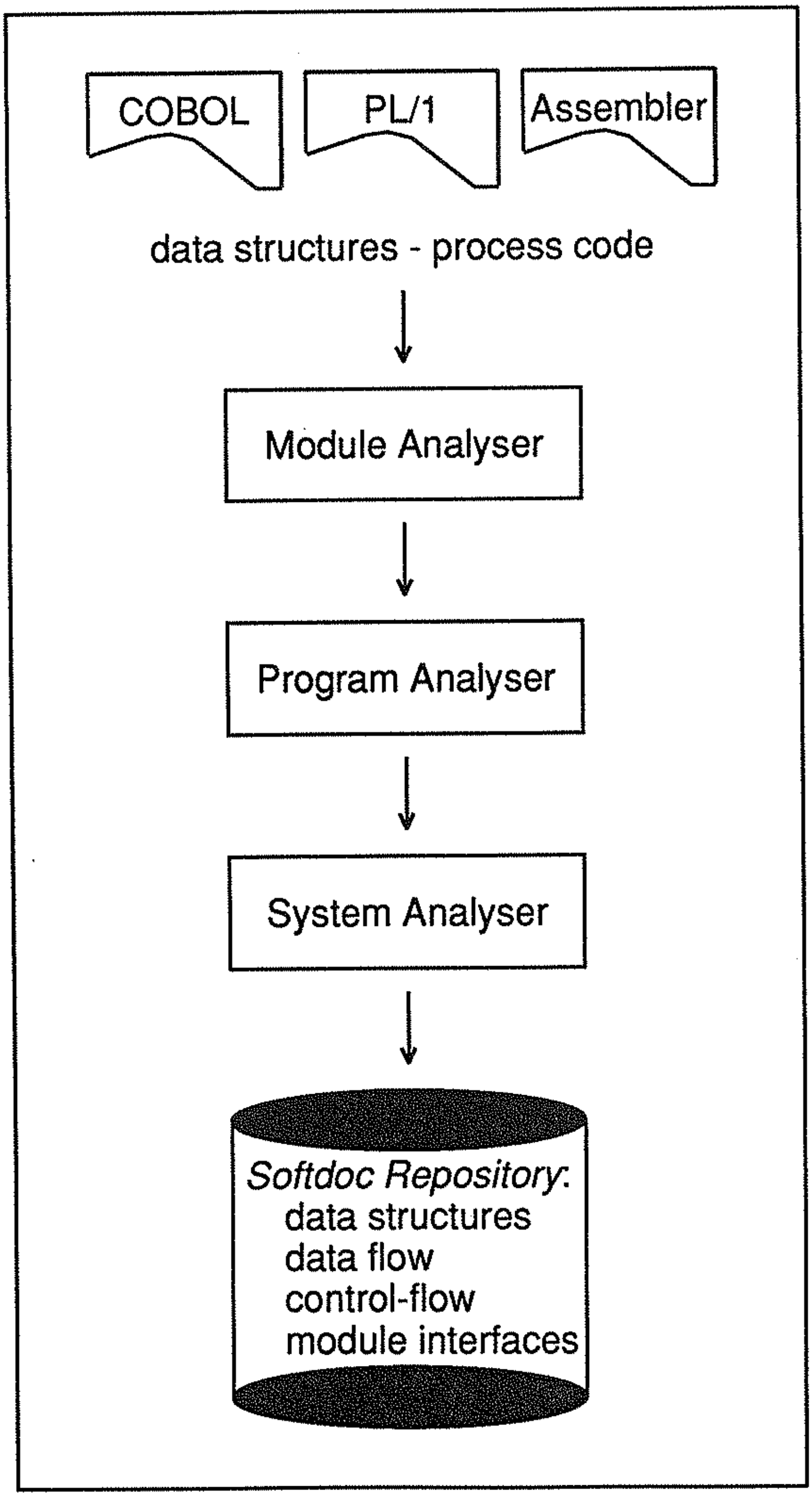
Developer:	Szamalk, Hungary
First issued:	1980
Target market:	IBM
Environment:	IBM mainframe
Price:	£4,800-£24,000 per module
Worldwide sales:	1988 - 50 sites

Softorg

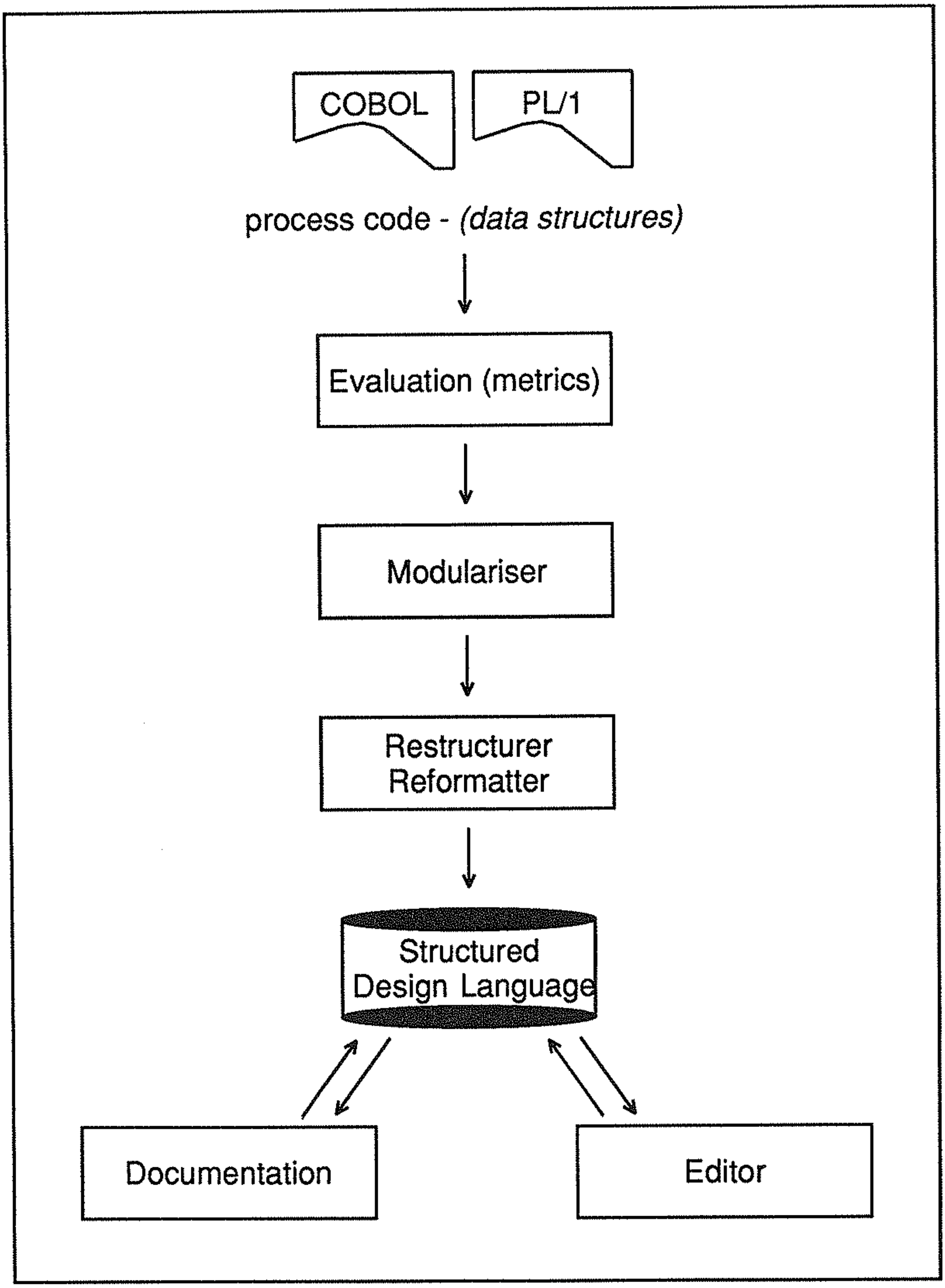
1. Project management
2. Strategy management
3. Forward engineering
4. *Reverse engineering: data and process code*

Softdoc:

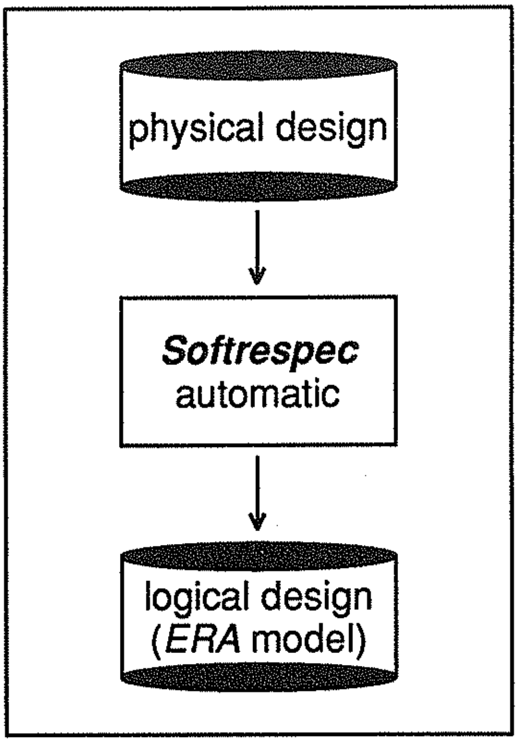
1. *RE* of data structures and program design
2. statistic report



Softreorg


COBOL
PL/1
process code - (data structures)
Evaluation (metrics)
Modulariser
Restructurer
Reformatter
Structured
Design Language
Documentation
Editor

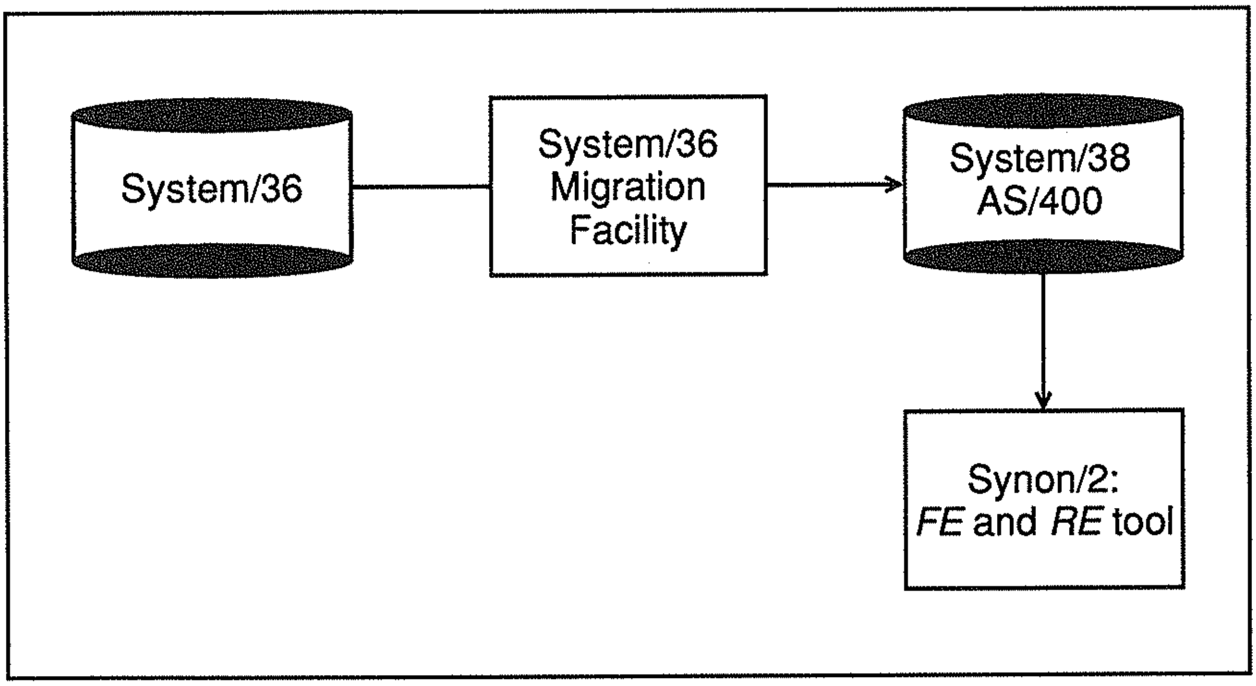
Softrespec

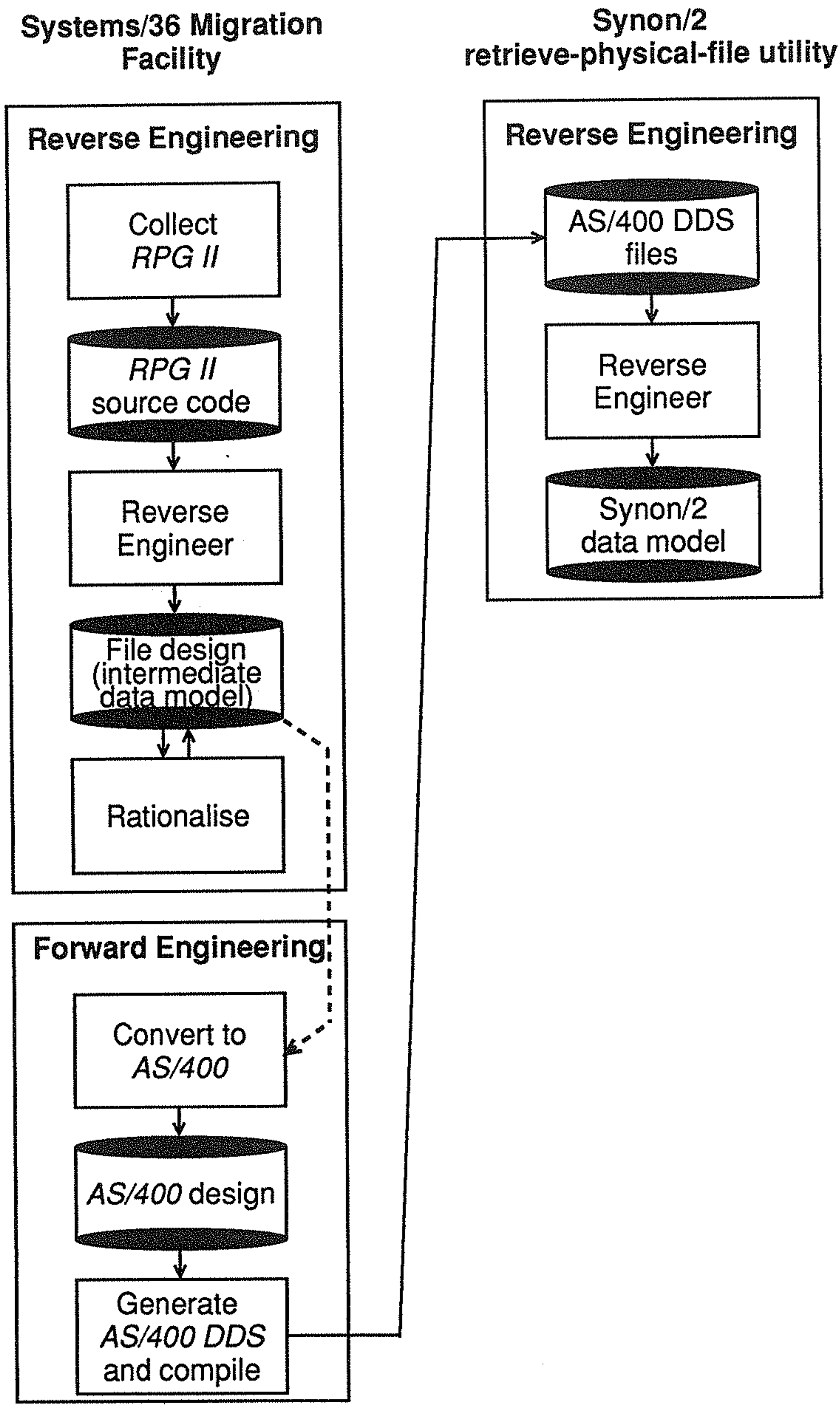

physical design
Softrespec
automatic
logical design
(ERA model)

Synon/2
System/36 Migration Facility

Developer:	Synon Inc, *USA*
First issued:	1986
Target market:	IBM System/36
	System/38
	AS/400 users
Environment:	IBM AS/400 and IBM System/38
Price:	£3,500 - £29,500 per CPU
Worldwide sales:	1989 - 1500 copies

Synon/2
System/36 Migration Facility



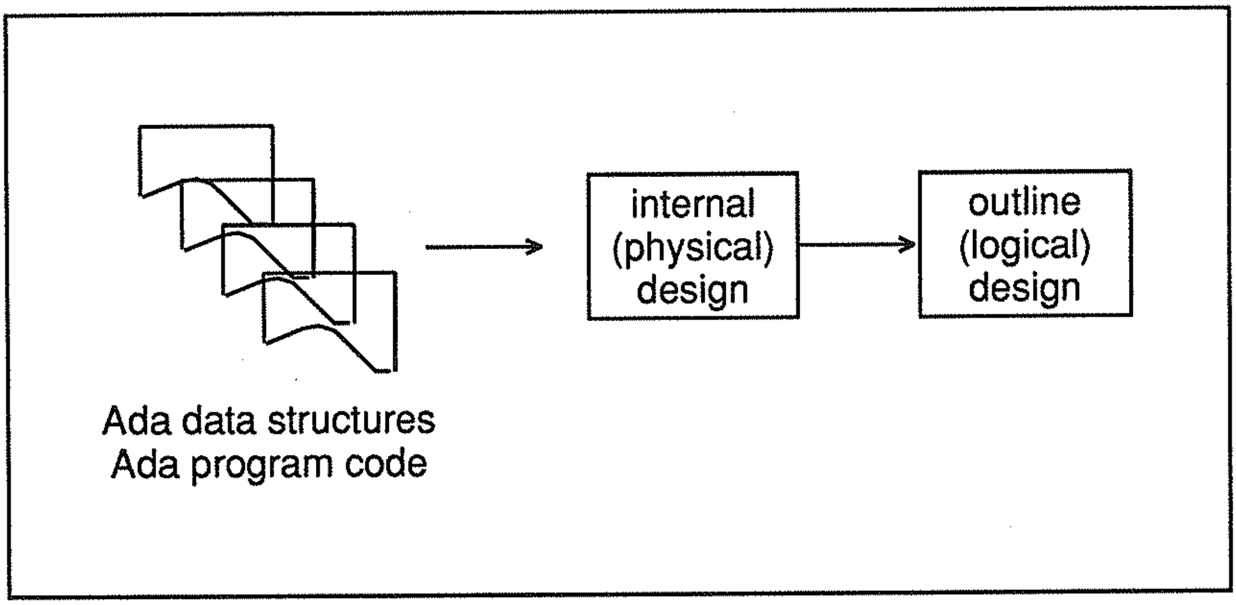

Systems/36 Migration Facility
Synon/2 retrieve-physical-file utility
Reverse Engineering
Collect RPG II
RPG II source code
Reverse Engineer
File design (intermediate data model)
Rationalise
Forward Engineering
Convert to AS/400
AS/400 design
Generate AS/400 DDS and compile
Reverse Engineering
AS/400 DDS files
Reverse Engineer
Synon/2 data model

AISLE

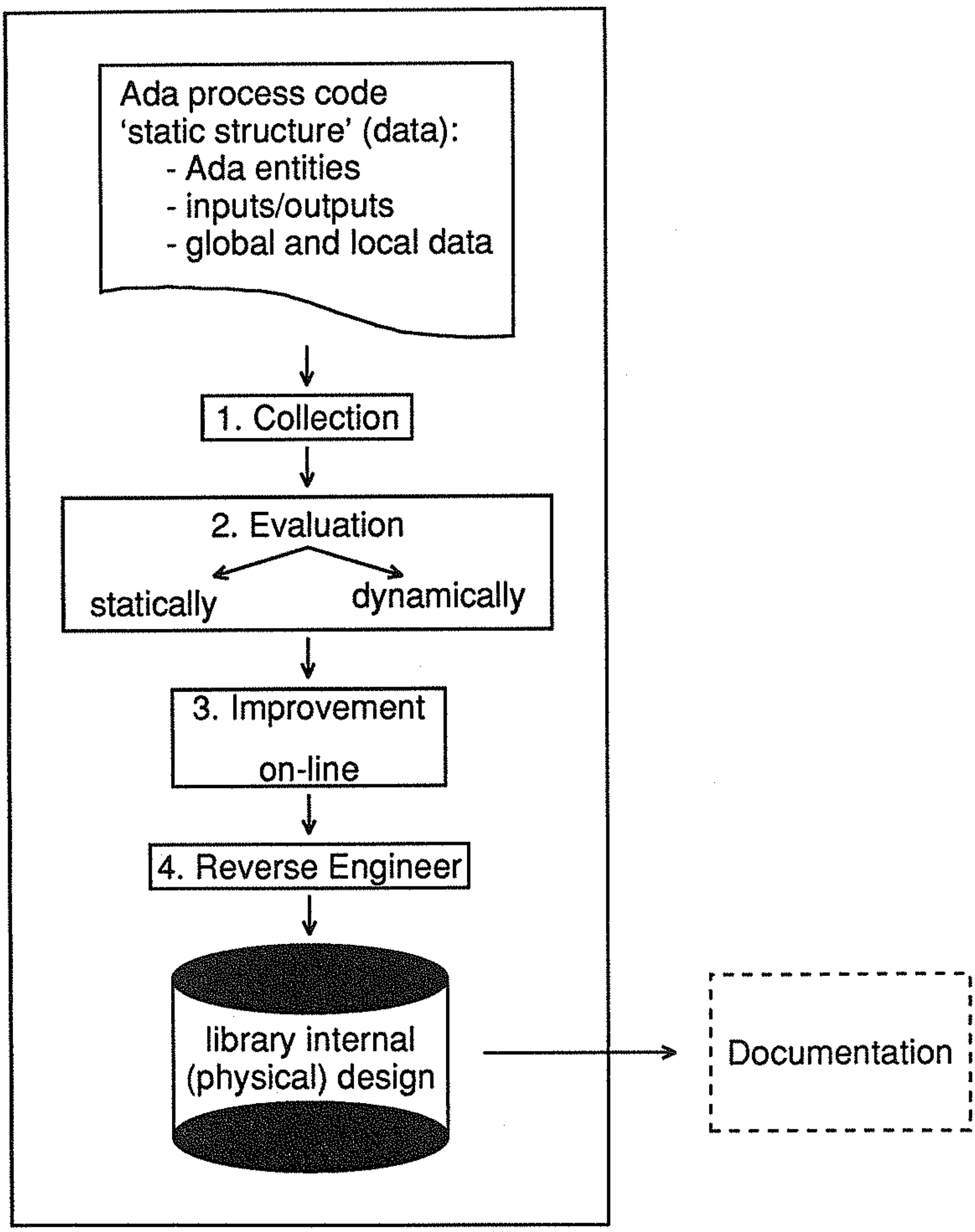
Developer:	Software Systems Design, *USA*
First issued:	1985
Target market:	Ada
Environment:	Various
Price:	$3,900 - $18,800 per CPU per module
Worldwide sales:	1989 - 45 sites

AISLE

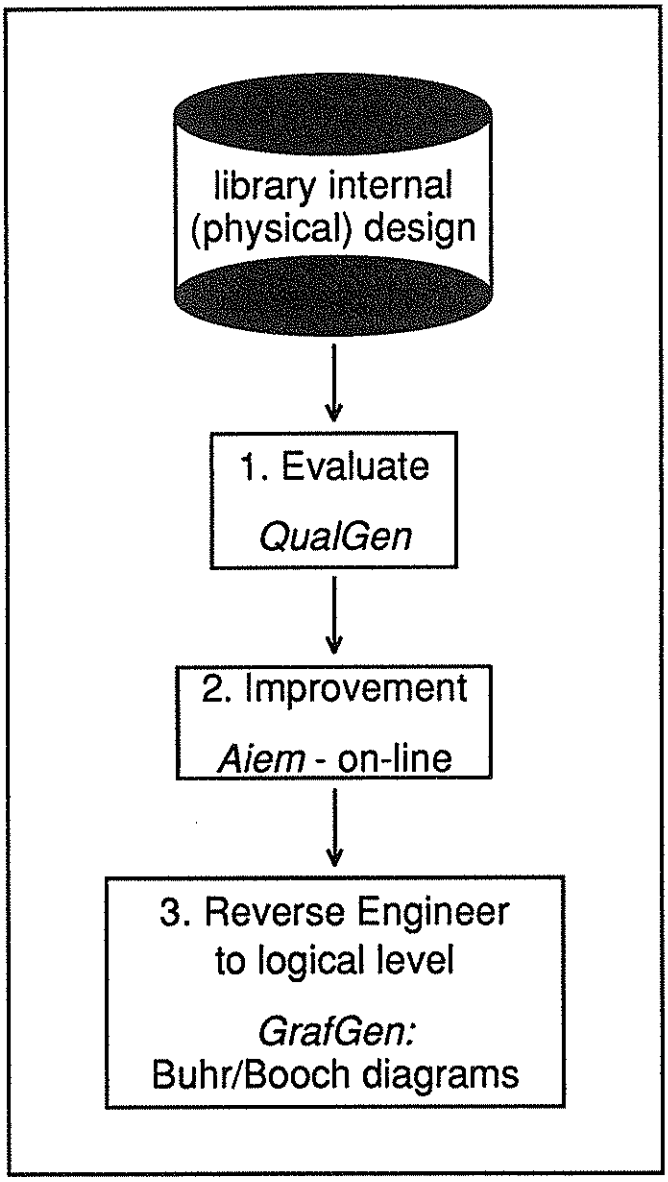
*A*da *I*ntegrated *S*oftware *L*ifecycle *E*nvironment



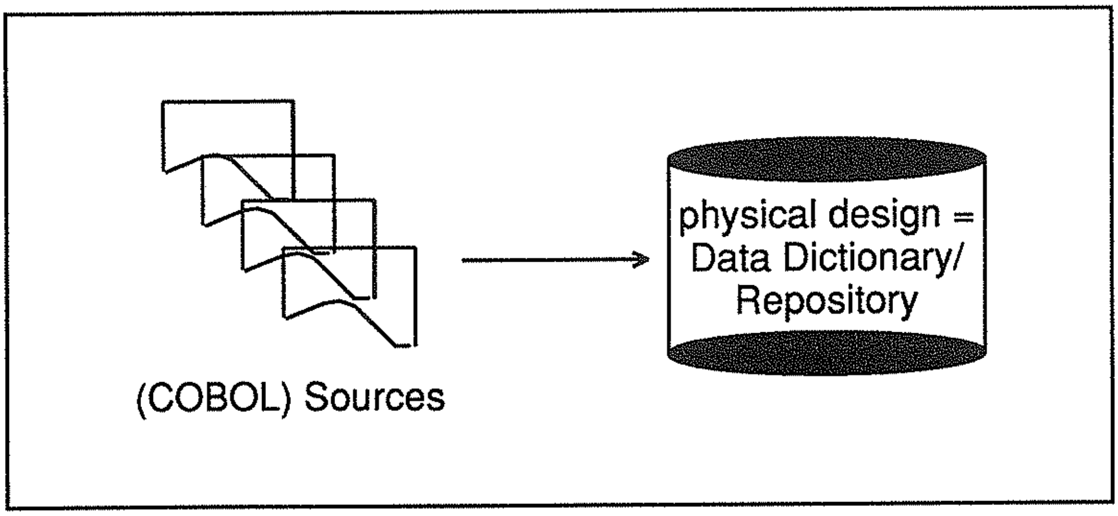
RE Ada code to an internal design



RE physical design to a logical design



Other *DBRE* tools



1. MSS Toolkit
2. PM/SS
3. PSL/PSA Reverse Engineering
4. PacBase Reverse Engineering
5. LCPS
6. Blues

SUMMARY

1. *DBRE* integrated in a *CASE* tool
2. automatic *RE* dominates
3. *DBRE* framework:
   a. reversal of *DB* forward engineering
   b. Bachman and Synon/2 reach conceptual level
4. Extraction:
   a. different source languages
   b. different *DBMS* types
   c. only data structures are extracted
   d. genericity ???
5. Central repository:
   a. kernel of the *DBRE* tool: Bachman
   b. genericity / language independent representation ???
6. Support for conceptualization: Bachman
7. Merging (integration): Bachman
8. Limited name processing functionality
9. Knowledge-based approach: Bachman

Phenix Project

Function / Tool	*Extraction*			*Central Repository*		*Conceptualization*	
	source languages	**DBMS**	**scope**	**kernel**	**genericity**	**level**	**support**
Redo	**COBOL**	**standard file**	**limited**	**yes**	**generic**	**conceptual**	**limited**
Bachman	**COBOL SQL**	**DB2 IMS IDMS std. file**	**limited**	**yes**		**conceptual**	**limited**
Synon/2	**RPGII AS/400**		**limited**	**no**	**no**	**logical**	**no**
Softorg	**COBOL PL/1 Assembler**		**complete**	**no**	**no**	**logical**	**no**
Aisle	**ADA**	**not relevant**	**complete**	**no**	**no**	**logical**	**no**
Phenix	**generic (COBOL)**	**standard file**	**complete**	**yes**	**generic**	**conceptual**	**yes**

Function / Tool	*Integration*	*Transformation*	*Name Processing*	*Interactive versus Automatic*	*CASE Integration*	*Knowledge Based*
Redo	no	limited	limited	interactive / automatic	yes	yes
Bachman	limited (merging)	limited	limited	automatic / interactive	yes	yes
Synon/2	no	limited	limited	automatic / interactive	yes	no
Softorg	no	limited	limited	automatic / interactive	yes	no
Aisle	no	limited	limited	automatic / interactive	yes	no
Phenix	yes	yes	yes	interactive / automatic	no	yes

Summary: Specificity of Phenix Project

1. Extraction:
    a. more than RECORD 01 definitions
    b. genericity
2. Object base = central repository of Phenix expert system: genericity
3. Support for conceptualization process:
    a. Integration
    b. Transformation
    c. Editors
4. Name processing
5. Interactive approach dominates
6. Knowledge-based approach: strategy module
7. Interesting *GUI* characteristics: e.g., Call graph
8. Origin management: maintain the link between original source and derived model
9. Knowledge extraction: experiments with real-world cases

Future Trends

1. Towards a maintenance methodology?
2. Future *CASE* tools will be a combination of:
    a. forward and reverse engineering
    b. data and process *RE*
3. Knowledge-based approach in *CASE* technology
4. *RE* dependent on the application domain
5. Limits of automated design recovery

Models, methods, and tools of DBRE: Introduction

Contents

Phenix Project:

1. Motivation and domain
2. Objectives
3. Methodology
4. Organization

Motivation and Domain of the Phenix Project

Motivation:

"reverse engineering as a solution for the maintenance crisis"

Domain:

- data base reverse engineering
- COBOL standard file systems

Objectives of the Phenix Project

1. Analyze the process of reverse engineering
2. Specify a *methodology* for *DBRE*
3. Develop a *RE* expert system

Project Development Methodology

Iterative development method:

1. Knowledge extraction: Experimentation (*case forms*)
2. Evaluation of results and formalization of the knowledge
3. Prototyping

Project Organization

1. Development environment:
   a. DECstation 3100 - ULTRIX/DECwindows
   b. Smeci - LeLisp - Aida - Masai expert shell environment

2. Two sites: Ghent and Namur

Main problems in database reverse engineering

Contents

- Introduction.
- A Generic Model.
- Data Structure Extraction <-> Physical Design.
  - -- DMS-DDL user's views or DMS-DDL global schema.
  - -- Physical Design.
- Data Structure Conceptualization <-> Logical Design.
  - -- Translation of a conceptual schema into a DMS compliant logical schema.
  - -- Optimization of the logical schema.
- Name analysis.

Introduction

Method :

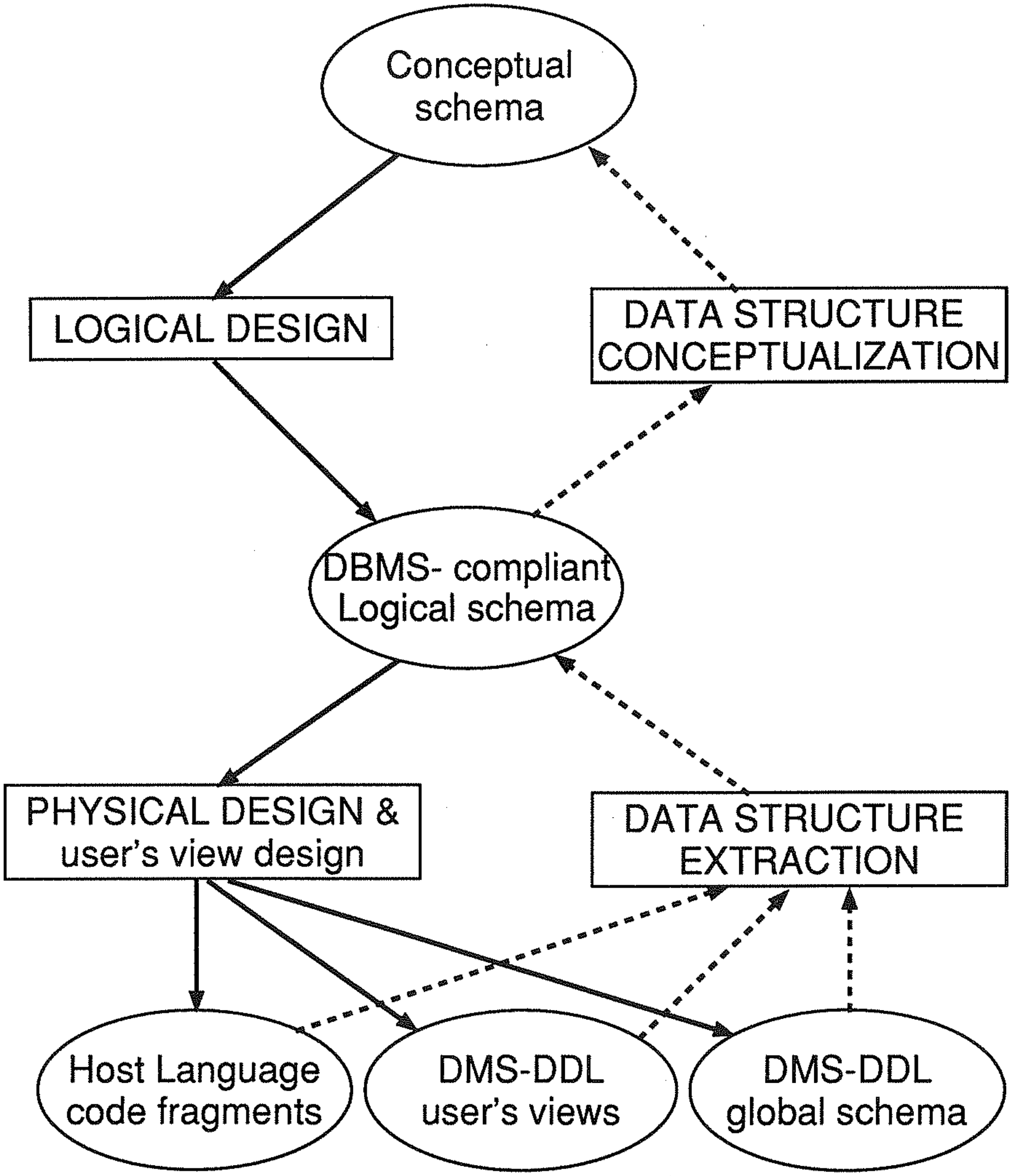
- Forward engineering <-> Reverse engineering.

Objectives :

- Forward engineering is a well known technique.
- View the reverse engineering process as the "reverse of forward engineering"
- Provide a generic framework for database reverse engineering.

Focus on problems encountered in reverse engineering.

A Generic Model



Data structure extraction <-> physical design.

definition :

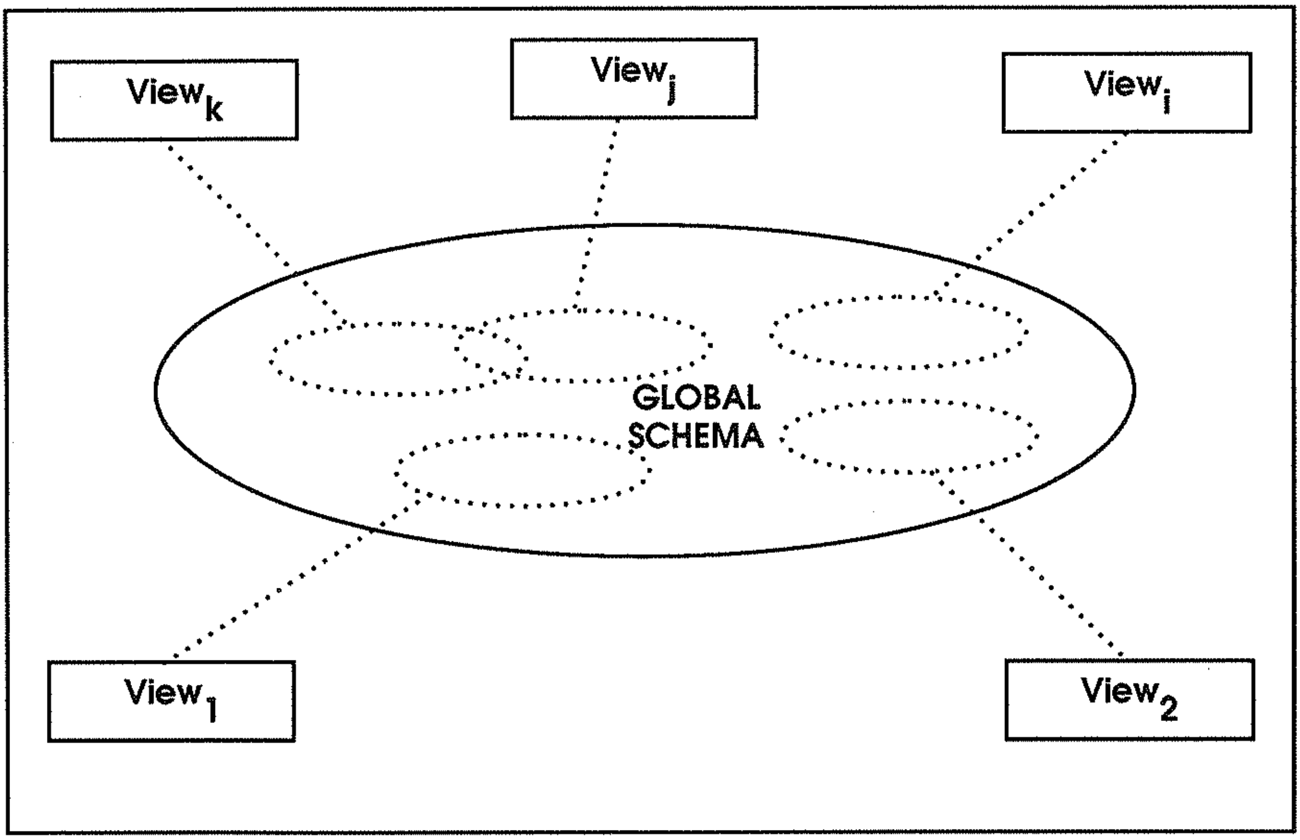
Data structure extraction is the process which generates the DBMS-compliant logical schema from Host Language code fragments, DMS-DDL user's views or from a DMS-DDL global schema.

DMS-DDL user's views or DMS-DDL global schema

- The choice is dependent on the type of DMS used.
- In advanced DMS systems the views are derived from a global schema.
  - -- The global schema is available in a DMS-DDL description, we can reverse engineer the global schema.
  - -- The DMS-DDL user views need not be reverse engineered.

- In simpler DMS systems (e.g. standard file systems) no global schema is available

  => ***multiple view problem.***



The global schema must be reconstructed from the different views.

This will give rise to several problems :

-- Find all sources which may include a user's view.

-- Extract those views from the sources.

-- The different schema's obtained can have following characteristics :

> *Schema redundancy* : a data structure is described in multiple views.

> *Syntactic variation* : the description of the same data structure can differ in syntactical description in the different views.

ex : different names, different structure , different length

```
CUSTOMER
  NAME X(10)
  BIRTH-DATE 9(6)
  ADDRESS
    STREET X(20)
    NUMBER 999
    CITY X(15)
```

```
CLIENT
  NAME
  FILLER 9(6)
  ADDRESS X(38)
```

> *Semantic variation* : The data can be perceived by the different views in a different way

  ex : The global employee data structure can be used for only male employees in one view and for only female employees in another.

> *Complementarity* : Some data structures are only described in some views and not in others.

- The technique which solves the multiple view problem is called schema integration or schema redundancy reduction.

Physical design

- The physical design step translates the optimized DMS-dependent logical schema into a DMS DDL schema.

PROBLEM :

Some logical concepts cannot be translated into DMS DDL concepts.

Physical parameters

- Normally physical parameter can be discarded.
- However sometimes they provide interesting information :
  - -- clustering information.
  - -- logical / physical file assignment.

Concepts from the DMS-DDL.

- The type of concepts which can be extracted is dependent on the DMS.

  ex :

IDMS	sets	-> relations.
IMS	hierarchical links	-> relations.
COBOL	records	-> entities.

- If the concept description in the DDL is complete extraction is an easy task.
- However sometimes the concepts are not completely described :

  ex :

  In *standard file systems* explicit description of records is not mandatory.

  In these situations the source must be analyzed to refine the descriptions.

  Detection and ***resolving*** gross descriptions is a difficult task to be done.

Concepts from non-DDL expressions.

- Concepts which cannot be expressed using the DDL language.

- Those concepts could be found in some procedural sections.

  ex :

  Referential integrity constraint in standard file systems

- It can be very difficult to retrieve those concepts.

  -- The concepts can be hidden.

  -- There are numerous ways of implementing those concepts

Data structure conceptualization <-> Logical design

- The logical design step translates the conceptual schema into an optimized DMS-dependent logical schema.

Translation of conceptual schema into DMS compliant logical schema.

Goal :

All non-DMS compliant concepts should be replaced by DMS compliant concepts.

Method :

Transformations (which are reversible) are used for this translation process.

Implications for reverse engineering :

- Which concepts derive from conceptual ones.

- Which type of transformation has been used.

Optimization of the logical schema

- Three types of optimization techniques do exist :
  -- Restructuring.
  Semantic preserving transformations.

  -- Denormalization (unnormalized structures).
    > This type of redundancy derives from non-key functional or multivalued dependencies.
    > They result in a redundancy at the instantiation level of the data structures.

ex :

CUSTOMER

NAME
ADDRESS
  STREET
  NUMBER
  CITY
  ZIP-CODE

CUSTOMER is a unnormalized structure since CITY and ZIP-CODE are mutually dependent.

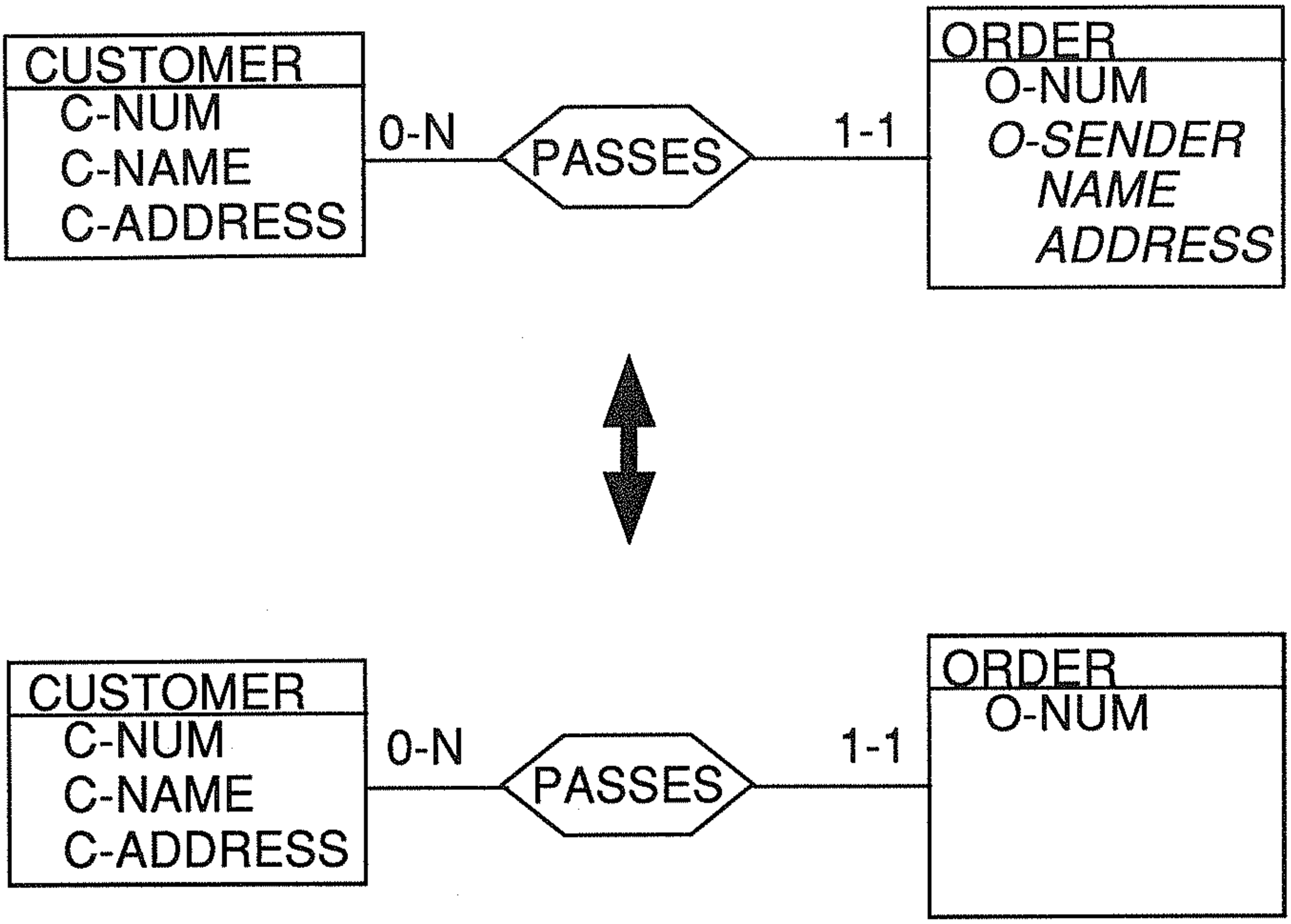
> Difficulty for reverse engineering : detection of the dependencies.

> Once unnormalized structures are known simple transformations allow for the normalization.

-- adding structural redundancies.

> Structural redundancy : structure A is redundant with structure B if all instances of A can be derived from all instances of B.

ex :



ORDER.O-SENDER.(NAME,ADDRESS)

copy-of ORDER.passes.CUSTOMER.(C-NAME, C_ADDRESS)

> Difficulty for reverse engineering : detection of the structural redundancies.

> Once detected they can be eliminated.

Name Analysis.

- Names of concepts can change during the forward engineering process.

- If no formal naming rules are applied following problems can occur during reverse engineering :

PROBLEMS ENCOUNTERED :

- *Synonyms* : the same concept can be described using several equivalent names.

  ex : CUSTOMER <-> CLIENT.

- *Homonyms* : the same name can be used to describe different concepts.

  ex : BAND -> narrow piece of material.

  -> group of musicians.

- *Language translations* : problems for applications running or developed in an international environment.

- *Abbreviations* : if abbreviations are too short then they are difficult to interprete.

  ex : C-NAME for CUSTOMER-NAME

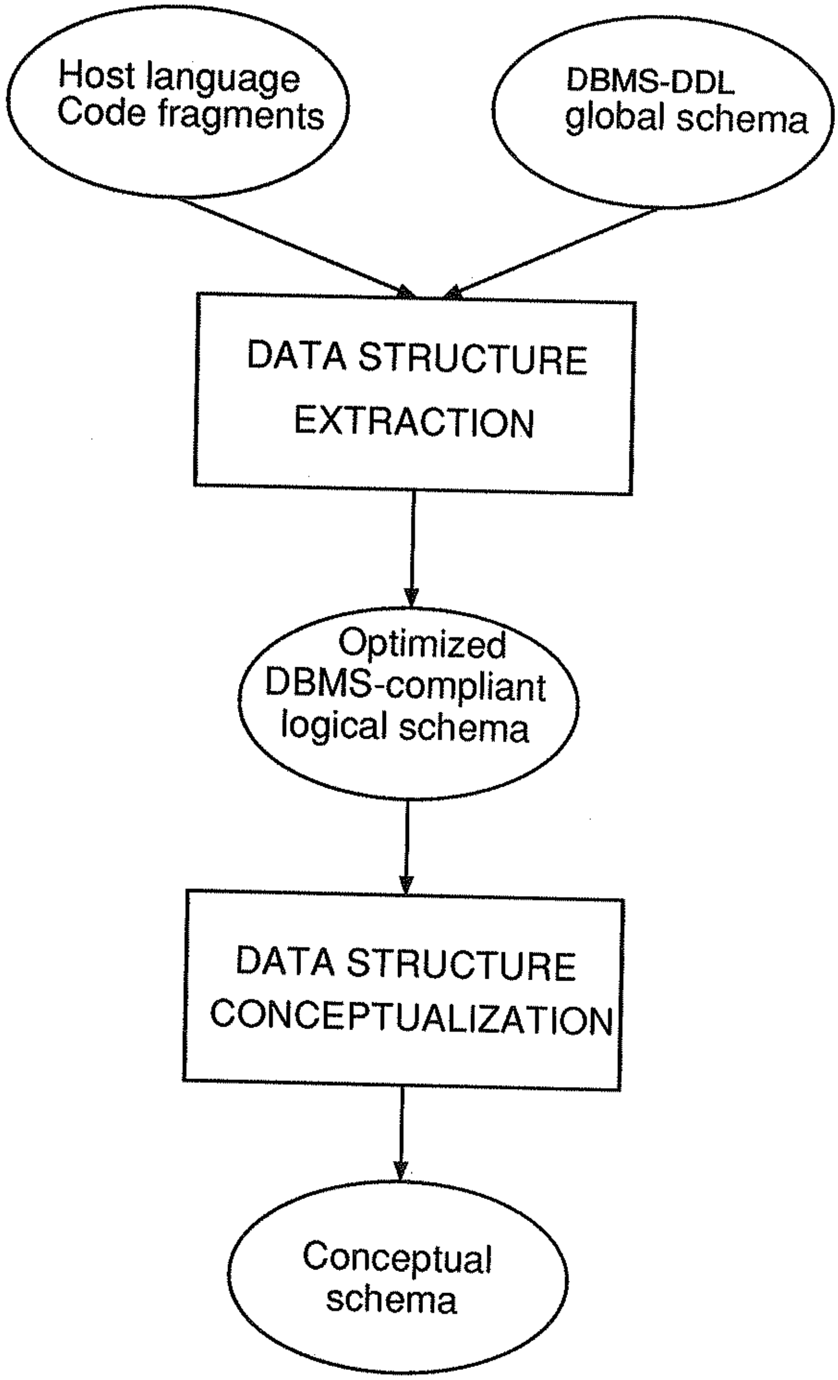
- *Context indications* : some names may include references to the context in which they are used :

  ex : CUSTOMER-NAME , PRODUCT-NAME

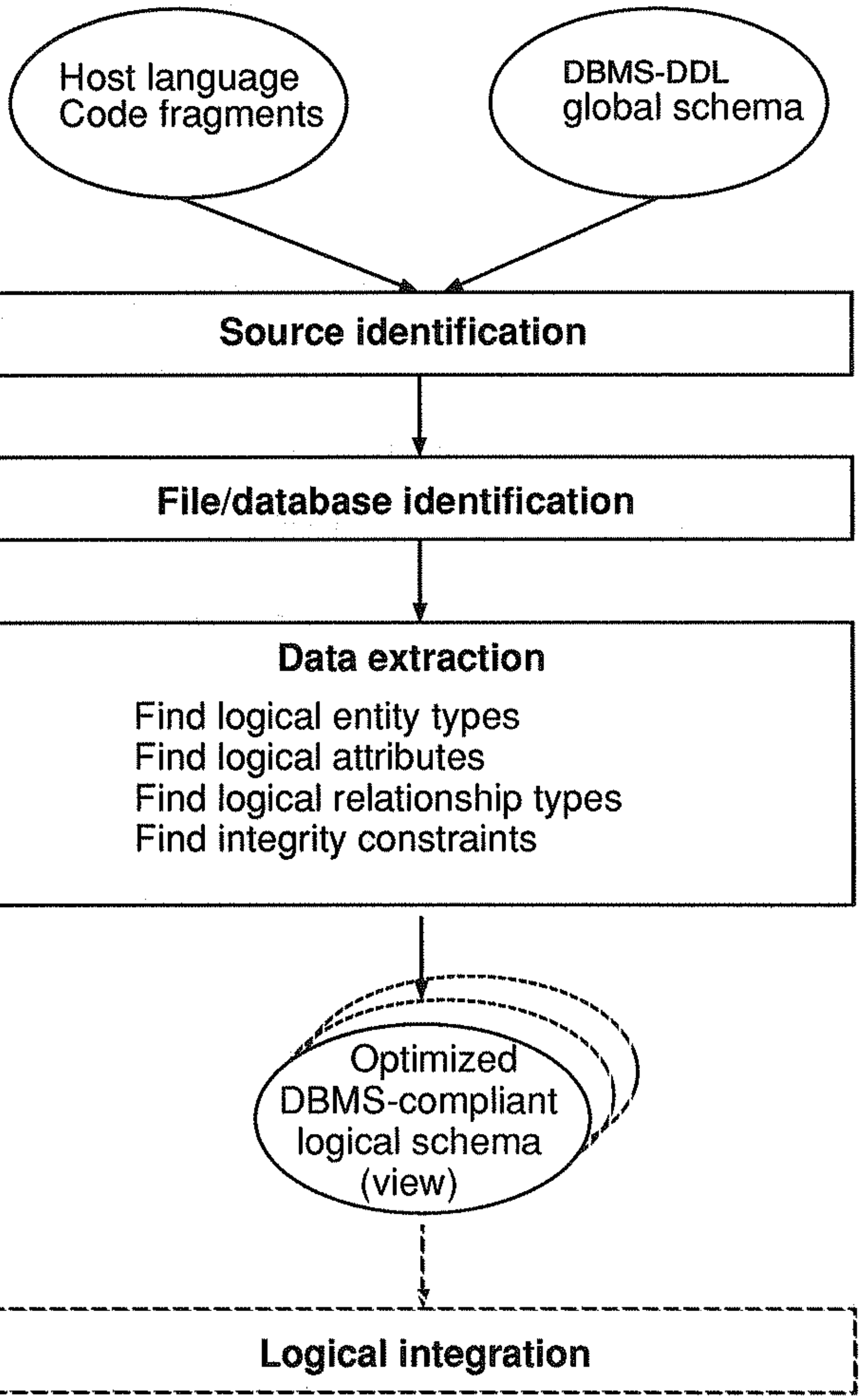
- *Incorrect spelling* : incorrect spelling can render difficulties in name analysis.

DBRE Methodology

A generic model for DBRE



Data structure Extraction phase



Data structure extraction phase

Sources identification

Choose the relevant text or data dictionary contents

Ordering sources of information in manageable packages

File/Database identification

Eliminate work, print, report, sort files

Data extraction

Analyze the source text in order to find out the DMS implementation of the logical structures

Analyze the source text in order to retrieve additional information on the logical data structures that have not be translated into the DMS/DDL.

Some descriptions are explicit, while others need a complex analysis of the procedural parts.

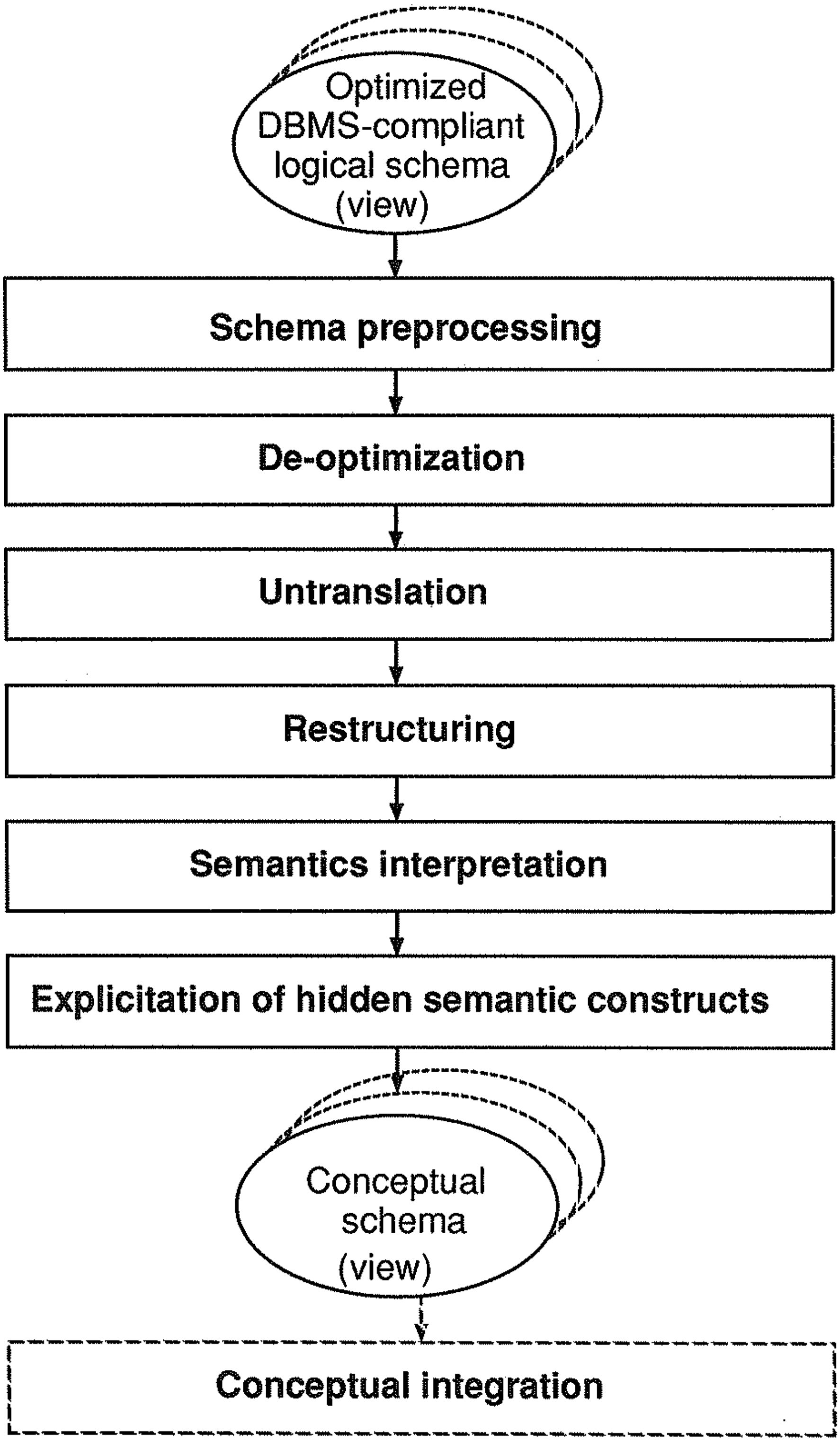
---> use extraction techniques

Logical integration

Find a unique global optimized-compliant schema by merging the several extracted user views

---> use integration techniques

Data structure conceptualization phase



Data structure conceptualization step

Schema preprocessing

Prepare the schema for further processing by making it more readable and eliminating obvious non-DB related constructs

---> use name processing, enrichment techniques

De-optimization

Remove technical constructs aimed at optimization (structural redundancy, technical redundancy, normalization, restructuration of optimized construct)

---> use transformation, enrichment techniques

Untranslation

Replacing constructs that have been introduced due to the limit of the expressive power of DMS model by more expressive ones.

---> use transformation techniques

Restructuring

Make the schema more readable, concise or compliant with corporate methodological standard

---> use transformation techniques

Semantic interpretation

Give an informal specification expressed in real world terms to data structures

---> enrichment techniques

Explicitation of hidden semantic constructs

Complete the data structure specifications and schemas with informations that have not be implemented due to internal limitations of the DMS/DDL

---> enrichment techniques

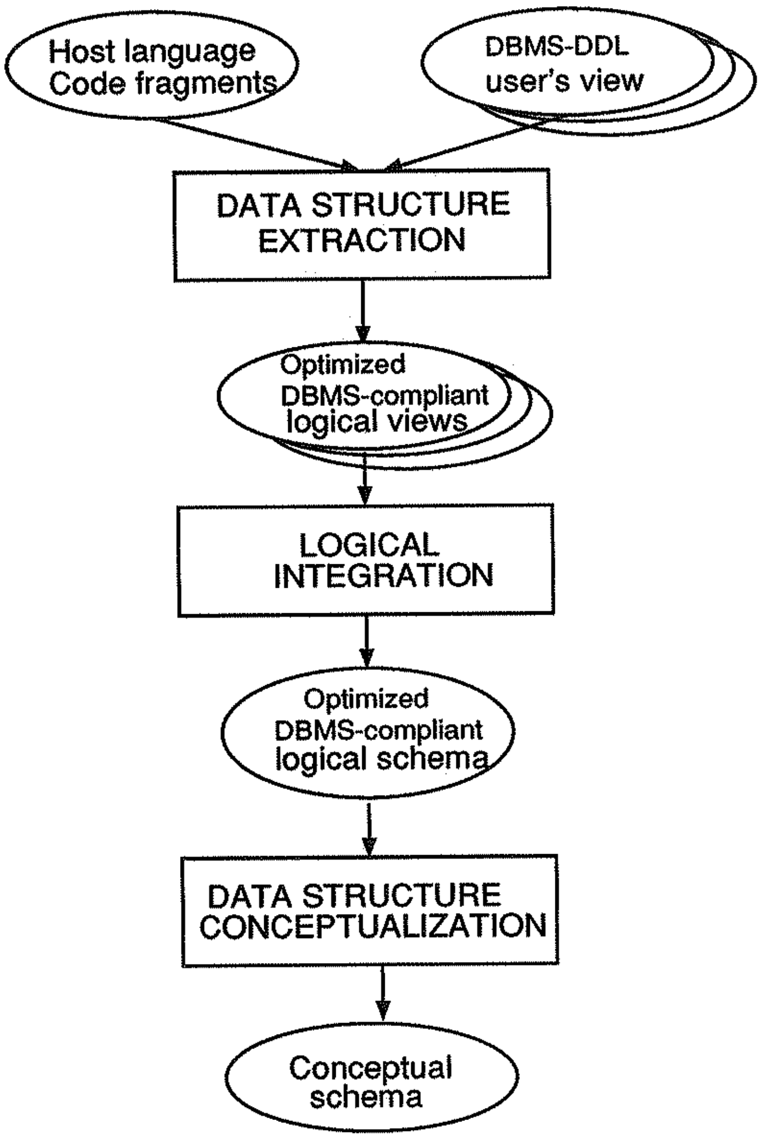
Integration

Find a unique global schema by merging the conceptual expression of the users views

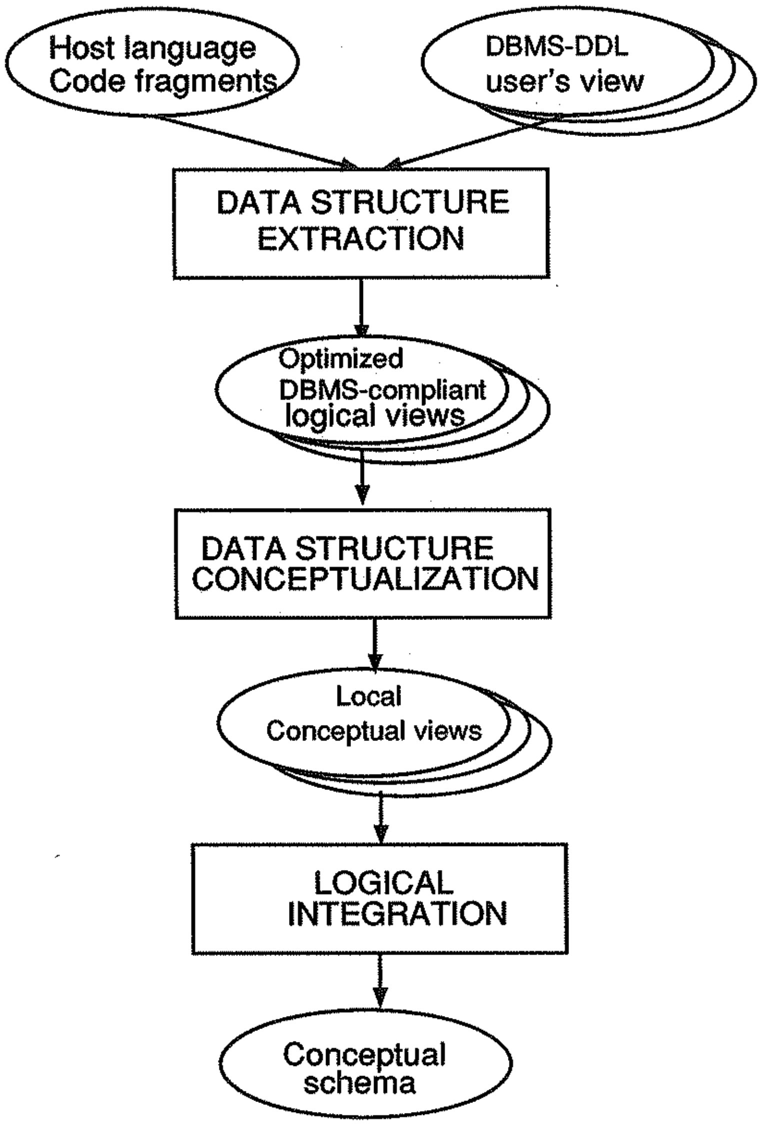
---> integration techniques

When integration can occur ?

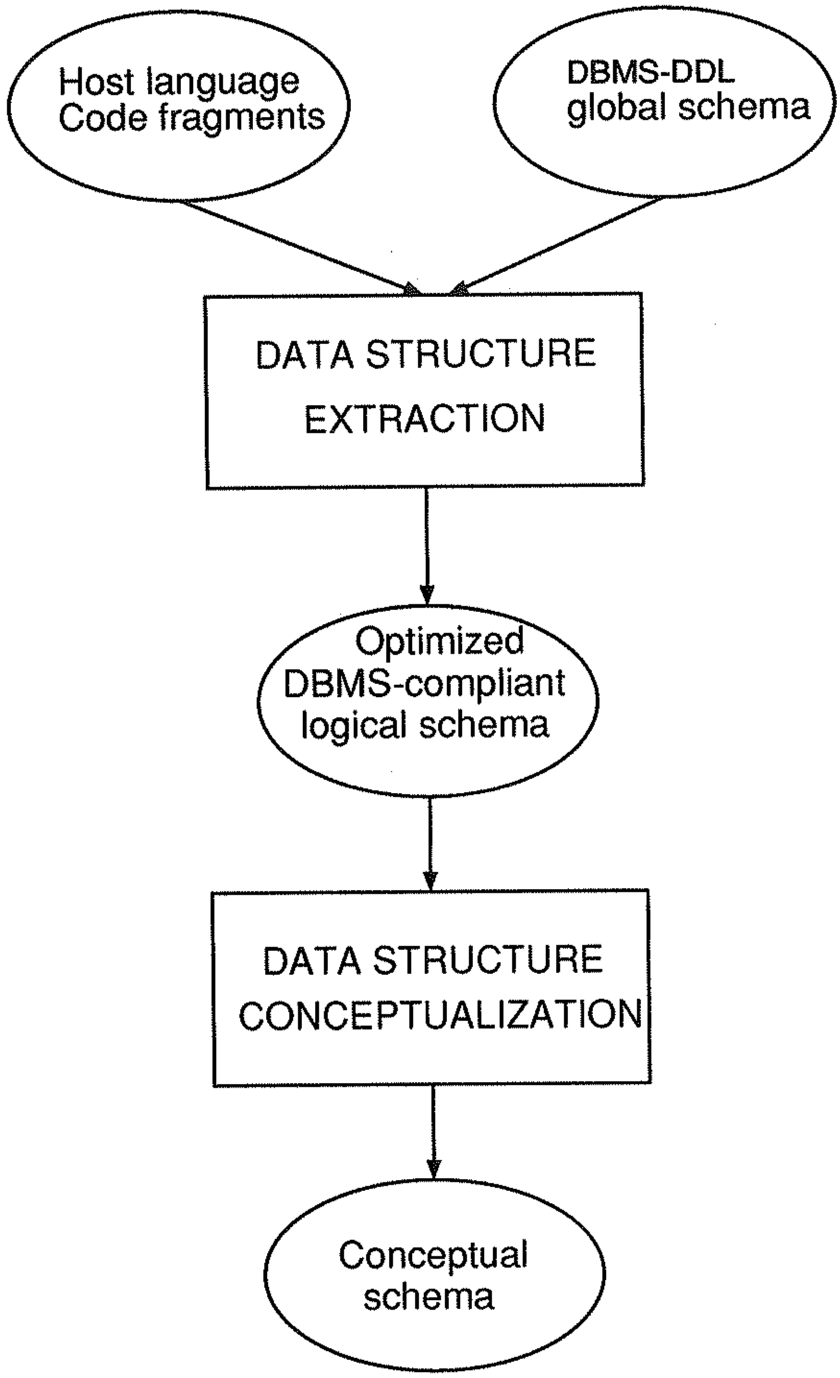
Early integration: low level DMS



Late integration: low level DMS



No-integration : High-level DMS



DBRE Basics techniques

Introduction

" Set of useful techniques whose combinations allow to solve problems arising in DBRE context "

- Some techniques are dedicated to a particular step of DBRE methodology.

e.g.: extraction techniques

- Some techniques are useful during all steps of DBRE methodology

e.g.: transformation, enrichment techniques

- Some techniques are well-known in FE and have been adapted or extended for RE.

Extraction Techniques

Definition:

"Set of techniques related to the retrieving of data structures and constraints in a particular host language (DMS and non-DMS parts)".

Used in:

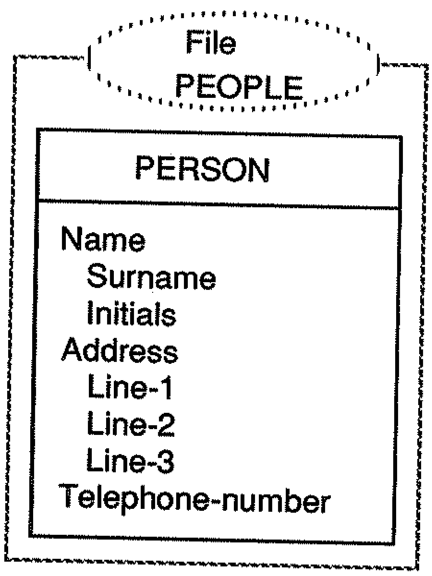
Extraction process: Reconstruction of the DBMS-compliant and optimized sub-schemas from a DMS-DDL/Host language

Examples:

- Find logical entity-types
- Find logical attributes
- Find logical relationship-types
- Find access keys
- Find identifiers
- Find referential constraint
- .....

Illustration

```
IDENTIFICATION DIVISION.
PROGRAM-ID. WRITE-TO-FILE.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.

          SELECT PEOPLE ASSIGN TO "A:PEOPLE.DAT"
          ORGANIZATION IS SEQUENTIAL
          ACCESS MODE IS SEQUENTIAL.

DATA DIVISION.

FILE SECTION.

FD PEOPLE.

01 PERSON.
          02 NAME.
                    03 SURNAME PIC X(10).
                    03 INITIALS    PIC A(4).
          02 ADDRESS.
                    03 LINE-1 PIC X(20).
                    03 LINE-2 PIC X(20).
                    03 LINE-3 PIC X(20).
          02 TELEPHONE-NUMBER PIC 9(10).
```



Integration Techniques

Definition:

"set of techniques related to the detection and the integration of equivalent data structures."

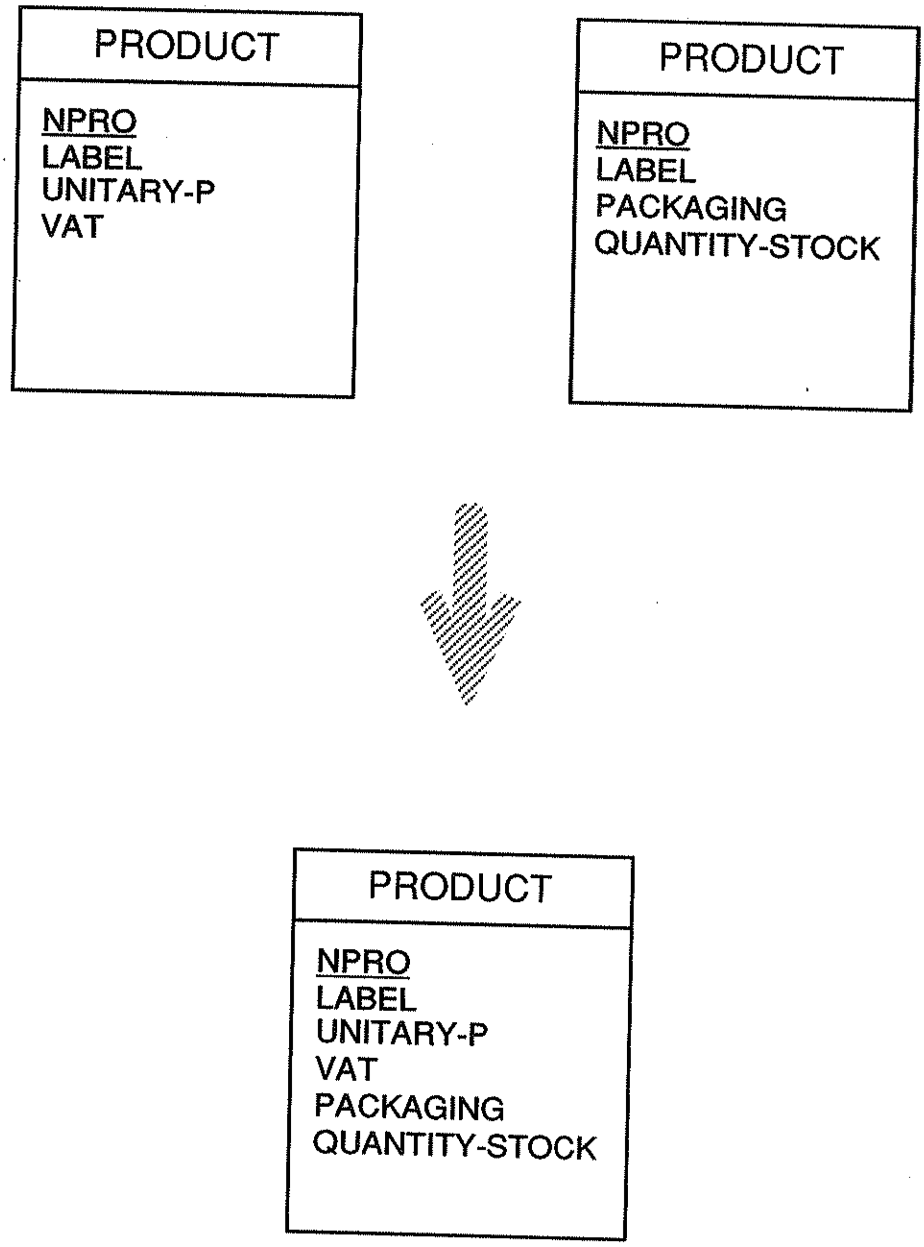
Used in:

- Internal logical integration: merging of extracted users views
- Internal conceptual integration: merging of conceptual users views
- Multibase integration: merging the concepts of several databases

Examples:

- Entity type/entity type integration
- Entity type/relationship type integration
- Entity-type/attribute integration
- Relationship type/referential attribute
- ....

Illustration



Transformation Techniques

Definition:

" A transformation consists in the replacement of a set of data structures by a semantically equivalent construct."

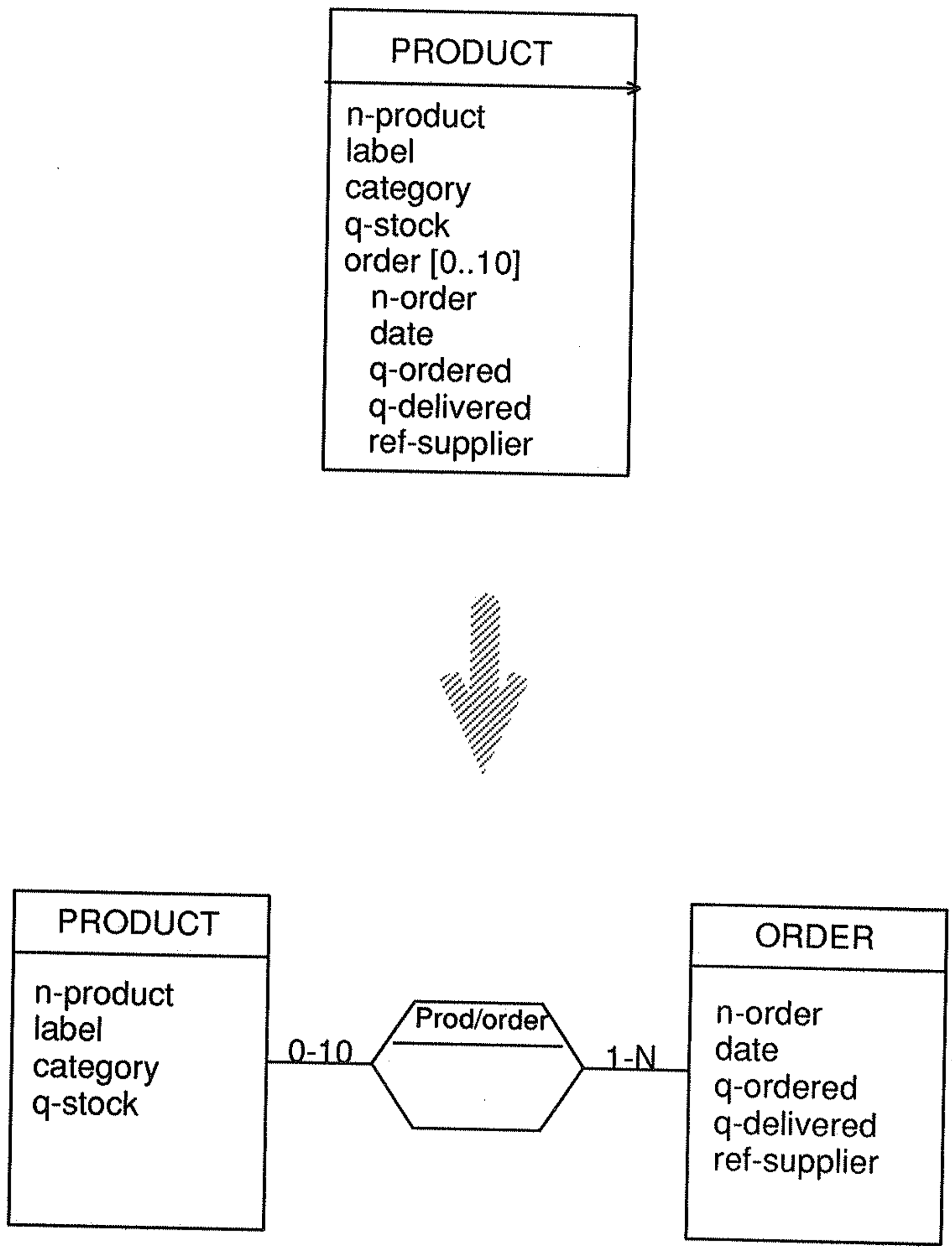
Used in:

- DMS de-optimization
- DMS untranslation
- Conceptual restructuring

Examples:

- Referential attribute -> relationship type
- Attribute -> entity type
- Aggregation of a list of attributes into a father attribute
- Specialization of an entity type
- ....

Illustration



Enrichment Techniques

Definition:

"Set of techniques which allow to modify the semantic of a construct by introduction, deletion or updating of data structures".

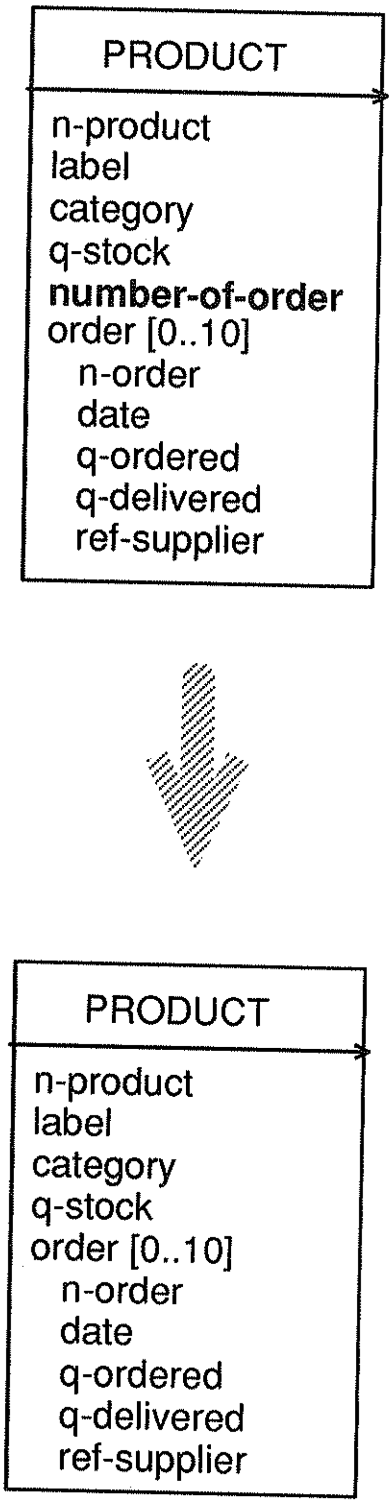
Used in:

- DMS de-optimisation
- Data redundancy elimination
- Semantic interpretation
- Explicitation of hidden semantic constructs
- Conceptual Enrichment

Examples:

- Creation of entity types, relationship types, is-a links, attributes,...
- Deletion of entity types, relationship types,...
- Updating of names, domains, semantic definition of concepts, ...

Illustration



Name Processing

Definition:

"Set of techniques related to name similarities detection and name conversions".

Used in:

- Preparation of the conceptualization step
- Detection techniques

Examples:

- Name truncation
- Name translation
- Name extension

Illustration

```
DATA DIVISION.
FILE SECTION.

FD Releves.

01 Releve-poste.
        02 Id-Poste.
                03 No-usine PIC is 99.
                03 No-atelier PIC is 99.
                03 No-poste PIC is 99.
        02 Date-releve.
                03 Annee PIC is 99.
                03 Mois PIC is 99.
        02 Heures-Releve.
                03 H-Normales PIC is 9(6).
                03 H-Suppl      PIC is 9(6).
```

Id ---> Identifiant
No ---> Numero
H ----> Heure
Suppl ---> Supplementaire

Annee ---> Annee du releve
Mois ----> Mois du releve

DBRE Strategy

Introduction

"Reverse engineering" is a "design" process
=> needs creativity.

"Reverse engineering is a "process"
=> needs of objectives to conduct the process.

Several techniques can solve a same problem
=> needs of decisions takings.

The physical schema is the result of an unknown FE design
=> needs of deep understanding of FE and RE processes and techniques.
=> needs of deep understanding of the context

There exist several hypothetical designs that should have lead to the same physical implementation.
=> needs to determine what are the final result(s) to obtain and the qualities of the result(s)

Why to reverse a file or a data base

What is it for ?

"To obtain an up-to-date documentation"

- understanding an application
- understanding an application domain
- validate existing documentation
- application restructuring
- application maintenance
- application conversion
- application integration

Who is it for ?

- BD designers
- programmers
- others ...

What are the qualities of the final schema

Faithful reflect of the application

As close as possible to the application reversed <->
As close as possible to the real world

Level of abstraction

Physical schema
Logical optimized and DMS-compliant schema or views
Logical DMS-independent schema
Conceptual schema

Compliance with a methodological standard

ERA, ERA-extended, binary, Niam, Bachman, ...

Multiple views <-> a unique schema

Clarity, conciseness, readableness

Where to find information

Sources texts

Data entry forms and reports

Existing documentation

Data dictionaries

Case tool

Database/files contents

Developer interview

User interview

Application domain

The Phenix tool

Contents

1. Introduction 3

2. Principles 4

3. Tool Overview 5

4. Editors 6

5. Tool functionalities 9

6. Methodological assistance 12

1. Introduction

Objectives

- to provide a tool which supports our DBRE models and methodology

Context

- development on two sites
- research context which strongly couples the development of the methodology and the tool
- prototyping strategy (two loops)

→ development time of the final product : ± 9 (wo)man/year

Agenda

- current state : under development
- final deadline : end 92

Programming Environment

- SMECI : Expert System Shell
- AIDA/MASAI : User Interface Development Tools

both based on LELISP

2. Principles

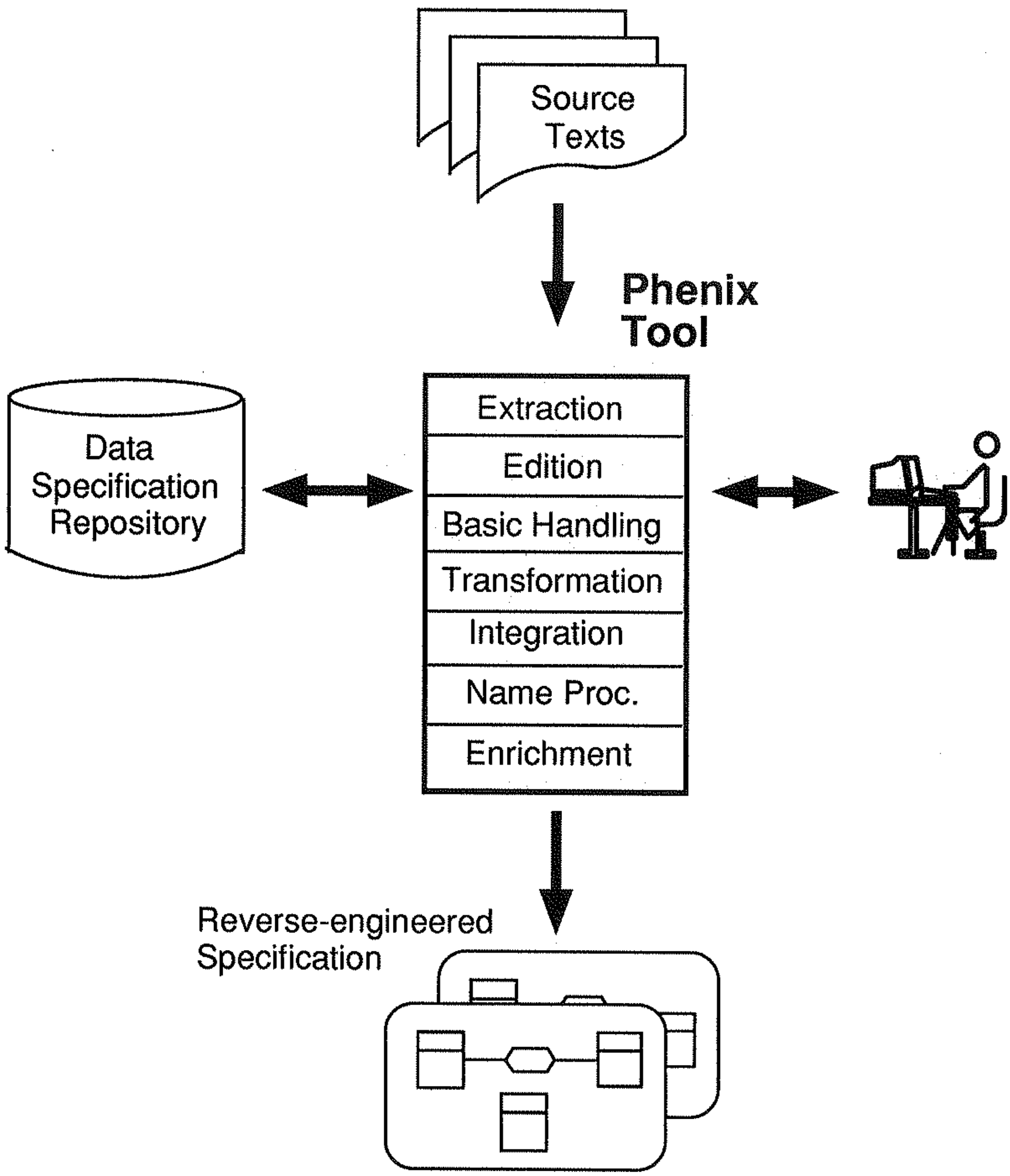
The priorities were placed on :

- Tackling the DBRE in its whole complexity
  → more than an automatic simplified tool
- Methodological concern
  → more than a toolbox (suggestions)

Conclusion : a *DBRE assistant*, dedicated to a DB reverse engineer

- Usability in a large context :
  - not too influenced by a specific DBMS;
  - meets the various goals of DBRE
- Mixing of expert system techniques/components with more classical ones :
  → Knowledge-Based System
- Not too expensive components :
  - reduced graphical user interface
  - not perfectly optimized performances
  - not perfectly optimized storage

3. Tool Overview



4. Editors

- **Application editor**

  shows its schemas, its global flow chart

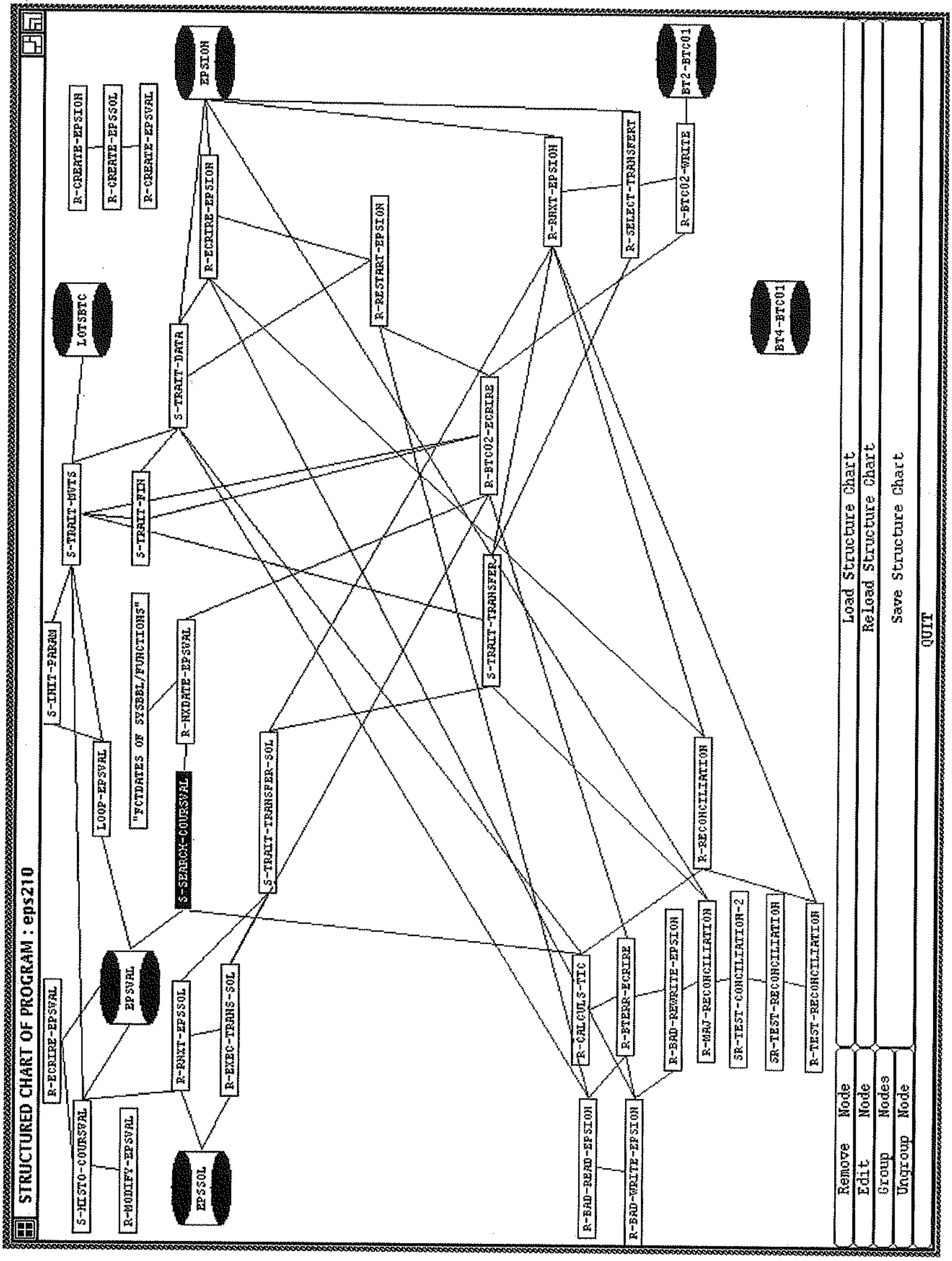
- **Schema editor** :

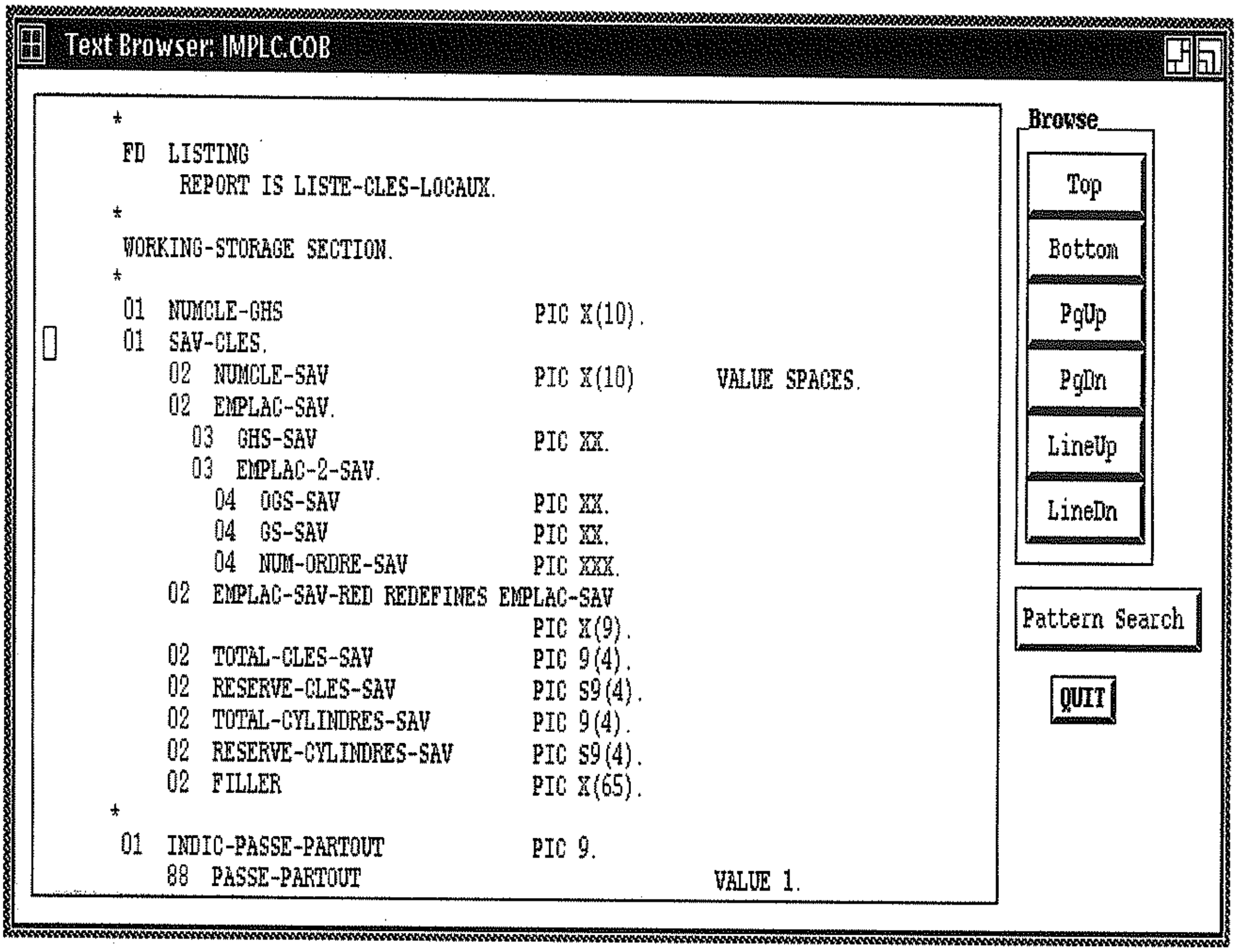
  shows :
  - its data structures (Entity/Relationship approach); allows the navigation in them.
  - its source files and their modules, its call graph.

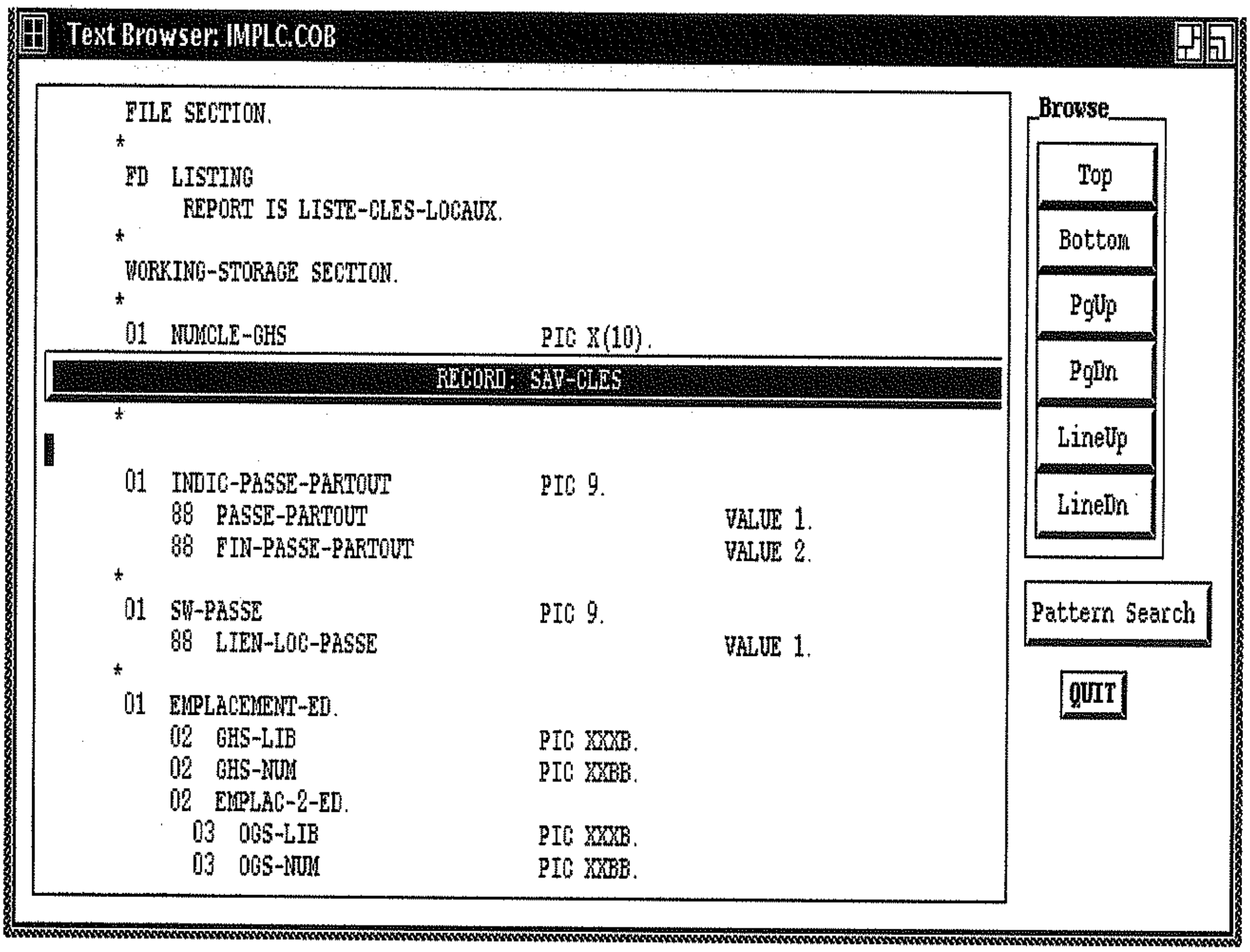
- **Data Structure editor** :
  shows its attributes, identifiers, (super/sub-types) access keys, sort keys, relative keys, logical files, physical files, constraints, comments.

- **Source Files editor** :
  shows (read only) a source text; allows the navigation in it; a search of patterns.
  Strong coupling with the data structure editor :
  - schema editor → source text editor : Where this structure comes from?
  - source text editor → schema editor : What is the structure which represents these lines?

- **Modules editor** :
  shows the calls to and from this module, its reports, its used logical files.


STRUCTURED CHART OF PROGRAM : eps210
R-CREATE-EPSION
R-CREATE-EPSSOL
R-CREATE-EPSVAL
EPSION
R-ECRIRE-EPSION
R-RESTART-EPSION
R-RNXT-EPSION
R-SELECT-TRANSFERT
R-BTC02-WRITE
BT2-BTC01
LOTSBTC
S-TRAIT-DATA
BT4-BTC01
S-TRAIT-MVTS
S-TRAIT-FIN
R-BTC02-ECRIRE
S-TRAIT-TRANSFER
S-INIT-PARAM
"FCTDATES OF SYSBBL/FUNCTIONS"
R-NXDATE-EPSVAL
LOOP-EPSVAL
S-SEARCH-COURSVAL
S-TRAIT-TRANSFER-SOL
R-RECONCILIATION
R-ECRIRE-EPSVAL
EPSVAL
R-RNXT-EPSSOL
R-EXEC-TRANS-SOL
S-HISTO-COURSVAL
R-MODIFY-EPSVAL
EPSSOL
R-CALCULS-TTC
R-BTERR-ECRIRE
R-BAD-REWRITE-EPSION
R-MAJ-RECONCILIATION
SR-TEST-CONCILIATION-2
SR-TEST-RECONCILIATION
R-TEST-RECONCILIATION
R-BAD-READ-EPSION
R-BAD-WRITE-EPSION
Load Structure Chart
Reload Structure Chart
Save Structure Chart
QUIT
Remove Node
Edit Node
Group Nodes
Ungroup Node


Text Browser: IMPLC.COB
*
FD LISTING
REPORT IS LISTE-CLES-LOCAUX.
*
WORKING-STORAGE SECTION.
*
01 NUMCLE-GHS PIC X(10).
01 SAV-CLES.
02 NUMCLE-SAV PIC X(10) VALUE SPACES.
02 EMPLAC-SAV.
03 GHS-SAV PIC XX.
03 EMPLAC-2-SAV.
04 OGS-SAV PIC XX.
04 GS-SAV PIC XX.
04 NUM-ORDRE-SAV PIC XXX.
02 EMPLAC-SAV-RED REDEFINES EMPLAC-SAV
PIC X(9).
02 TOTAL-CLES-SAV PIC 9(4).
02 RESERVE-CLES-SAV PIC S9(4).
02 TOTAL-CYLINDRES-SAV PIC 9(4).
02 RESERVE-CYLINDRES-SAV PIC S9(4).
02 FILLER PIC X(65).
*
01 INDIC-PASSE-PARTOUT PIC 9.
88 PASSE-PARTOUT VALUE 1.
Browse
Top
Bottom
PgUp
PgDn
LineUp
LineDn
Pattern Search
QUIT


Text Browser: IMPLC.COB
FILE SECTION.
*
FD LISTING
REPORT IS LISTE-CLES-LOCAUX.
*
WORKING-STORAGE SECTION.
*
01 NUMCLE-GHS PIC X(10).
RECORD: SAV-CLES
*
01 INDIC-PASSE-PARTOUT PIC 9.
88 PASSE-PARTOUT VALUE 1.
88 FIN-PASSE-PARTOUT VALUE 2.
*
01 SW-PASSE PIC 9.
88 LIEN-LOC-PASSE VALUE 1.
*
01 EMPLACEMENT-ED.
02 GHS-LIB PIC XXXB.
02 GHS-NUM PIC XXBB.
02 EMPLAC-2-ED.
03 OGS-LIB PIC XXXB.
03 OGS-NUM PIC XXBB.
Browse
Top
Bottom
PgUp
PgDn
LineUp
LineDn
Pattern Search
QUIT

5. Tool functionalities

5.1 Extraction

Three levels of granularity are provided :

Main extraction :

- the user is assisted in its choice of the most pertinent source files, to group them into schemas by :
    - checking consistency (calling/called, input/output)
    - showing statistics
- the extraction of basic concepts is then performed automatically

Incremental extraction :

- when additional informations on a given structure are required, a local extraction process is carried out

Manual extraction :

- the user can create additional concepts (constraints e.g.), guided by a pattern research in source texts

5.2 Integration

- Two granularity levels :
  - integration of two schemas
  - integration of two basic structures
- Three suggestion levels :
  - the user knows the two structures to be integrated
  - the user gives a structure and the tool suggests corresponding structures to be integrated
  - the tool suggests couples of structures to be integrated
- Integration is processed in a half-automatic way

5.3 Transformation

- A set of 20 transformations which allow to obtain more conceptual structures are available to the user.
- The transformations are performed fully automatically.
- In suggestion mode, the tool can give hints for applying them

5.4 Name processing

- The user can remove prefixes or suffixes in a (set of) data structure(s)
- The user can perform string replacement in a (set of) data structure(s)
- The system can suggest a correct name for a new data structure (useful during integration or transformation)

5.5 Basic handling

- The user has at his disposal "surgery" operations for creating/modifying/deleting any element of data specifications

5.6 Enrichment

- The user can perform semantical enrichment processes, such as creating a whole data structure, defining an is-a hierarchy on existing data structures, etc.

6. Methodological assistance

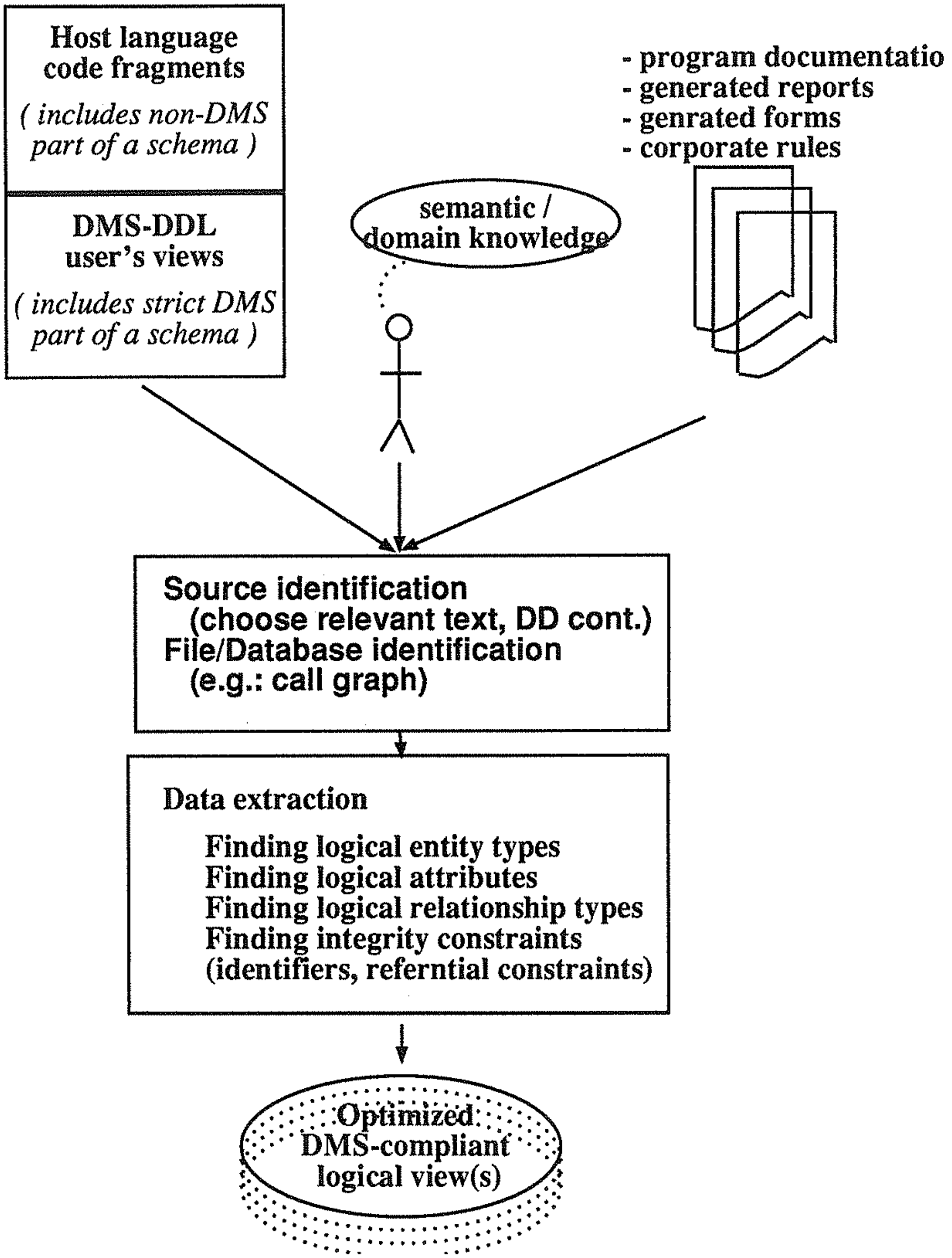
- **Suggestion mode** which can be switched (on/off) based on both :
  - a static knowledge about RE methodologies (strategical knowledge)
  - a dynamic analysis of the current state of specification

- **Versioning** capabilities :
  - Save / Save as
  - Schema State-tree editor, during a session, allowing:
    - multiple undo
    - multi-hypothesis

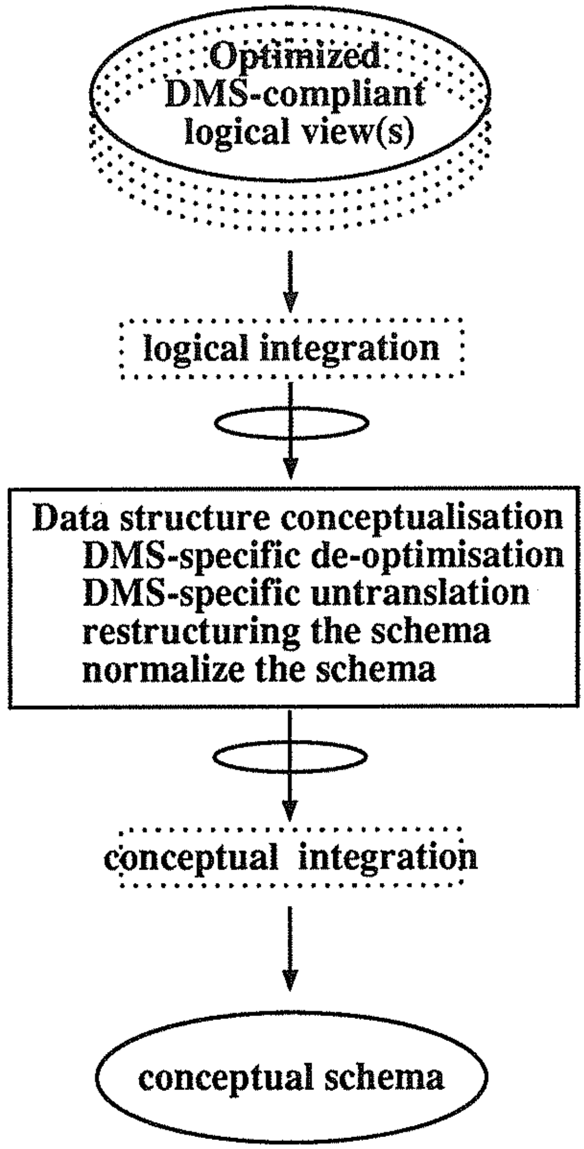
Case study

A database of standard COBOL files

Generic model for reverse engineering (part 1):



Generic model for reverse engineering (part 2) :



PART 1 : SOURCE ORGANIZATION,
WORK ORGANIZATION.

Step a :

source identification,

source grouping,

composition of the application,

....

- non-automized part:

  collect documentation about application (if available),

  collect information about used methodology during development of the application (if available)

  collect sources of application

  .....

- automized-part :

  inspect collected sources: statements :

  > PROCEDURE USING ...

  > CALL ... USING ...

  > READ ....

  > WRITE ...

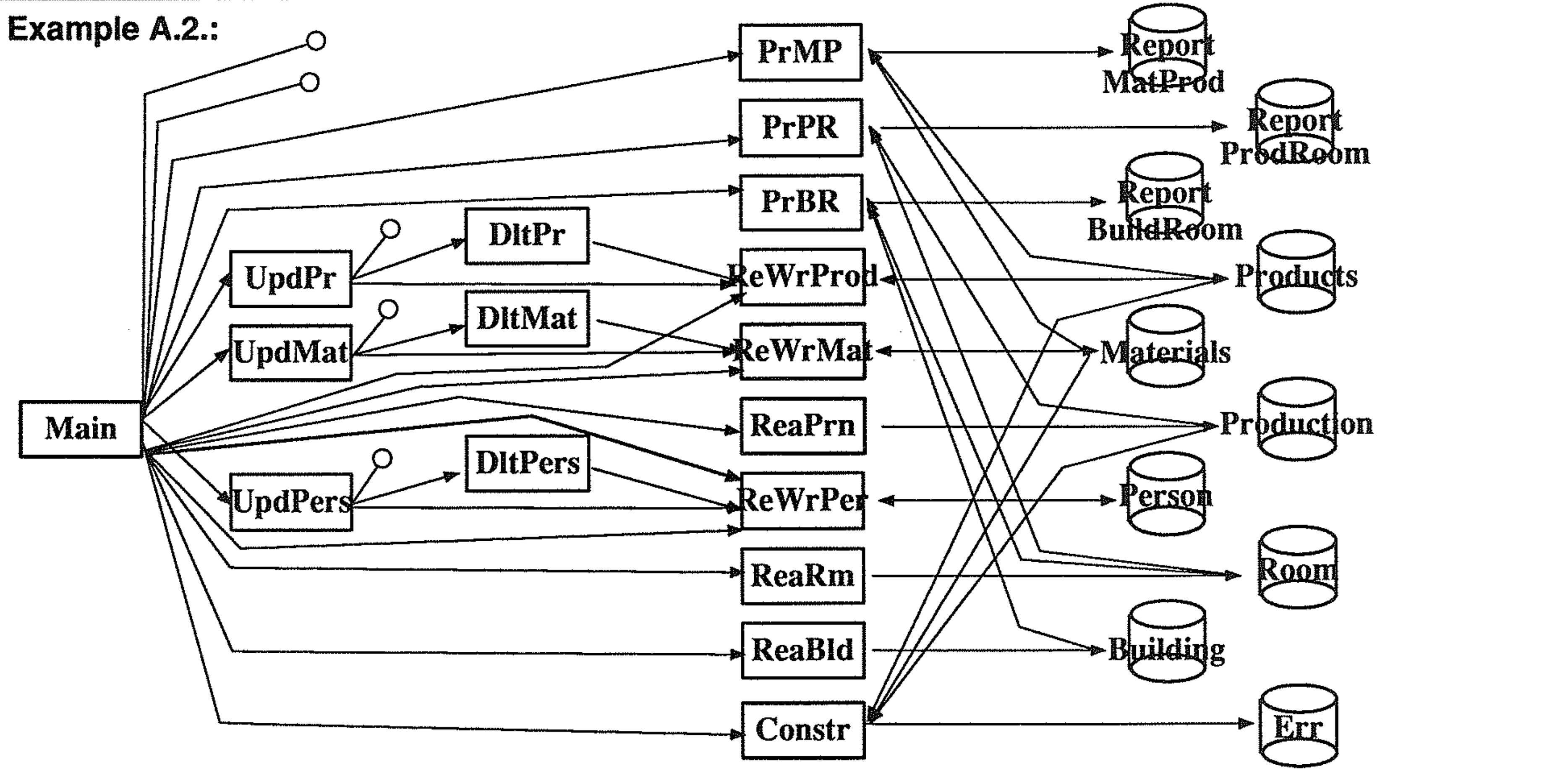
  > GENERATE ...

Example A.1.:

```
PROCEDURE DIVISION USING
        ART-FICHIER-LI ART-LOCAL-LI ART-CLES-LI
        INDIC-FICHIER-LI CHEMIN-LI FILE-STATUS-LI.

    IF PREMIER-APPEL

        MOVE 1 TO INDIC-CREATION

.....

LECTURE-RANDOM-PERCLE.
    READ PERCLE INVALID KEY
      DISPLAY  " **** ERREUR DE LECTURE DANS PERCLE.RAN - "
               "RELATIVE KEY = "  RELKEY-PERCLE
      MOVE 1 TO INDIC-ERREUR-GRAVE.

LECTURE-RANDOM-LOCCLE.
    READ LOCCLE INVALID KEY
      DISPLAY  " **** ERREUR DE LECTURE DANS LOCCLE.RAN - "
               "RELATIVE KEY = " RELKEY-LOCCLE
      MOVE 1 TO INDIC-ERREUR-GRAVE.
 .......

APPEL-MODULE-LECPER.
      CALL "LECPER" USING

              PARAMETRES-APPEL-LECPER-WO

              ART-PERSO-WO INDIC-PERSO-WO.

APPEL-MODULE-LECCLE.
      CALL "LECCLE" USING

              PARAMETRES-APPEL-LECCLE-WO

              ART-CLES-WO INDIC-CLES-WO.
```

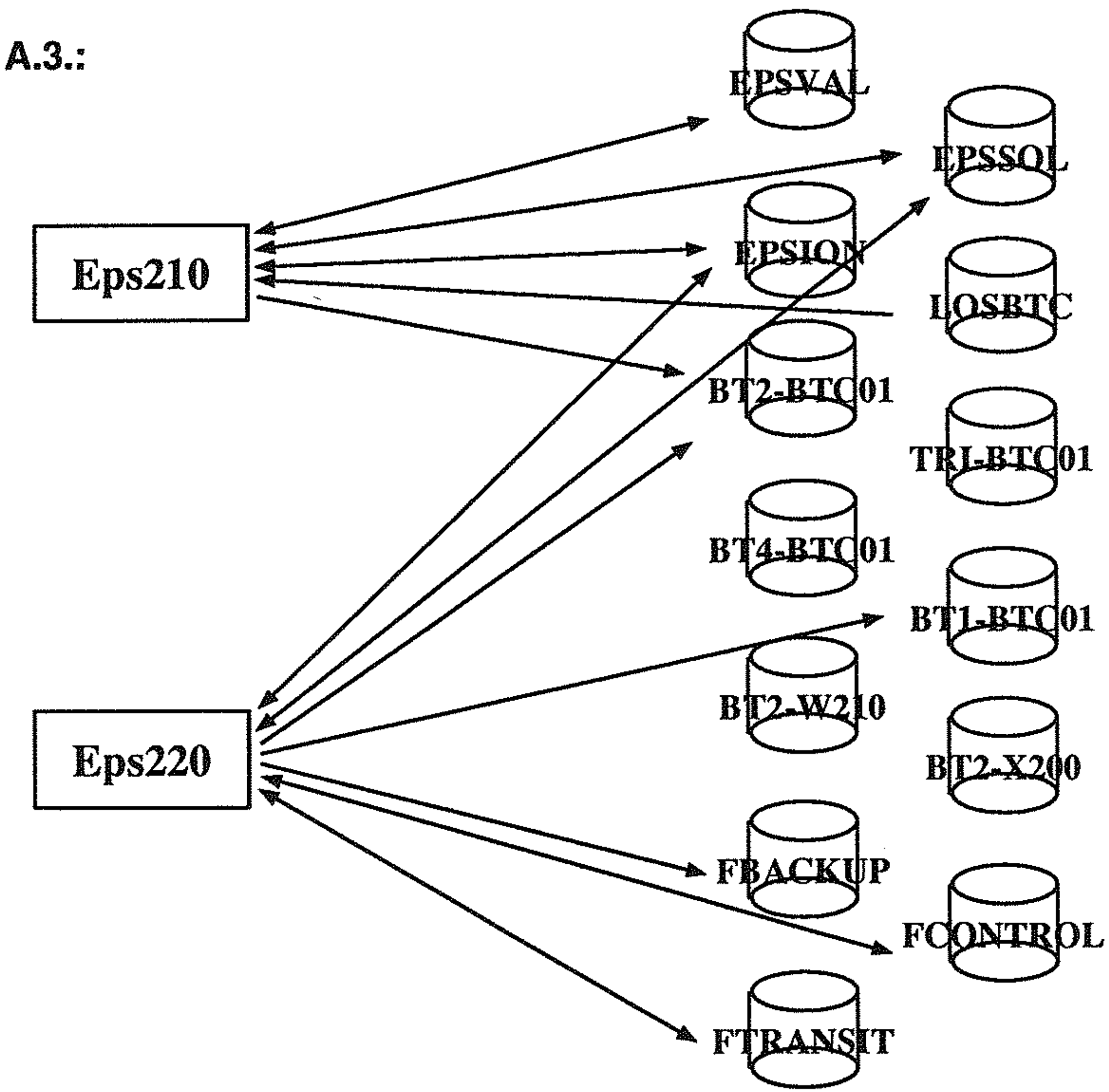
==> call graph

Example A.2.:



example of a selection of sources to create a partial schema

Example A.3.:


Eps210
Eps220
EPSVAL
EPSSOL
EPSION
LOSBTC
BT2-BTC01
TRI-BTC01
BT4-BTC01
BT1-BTC01
BT2-W210
BT2-X200
FBACKUP
FCONTROL
FTRANSIT

Step b:

construction of conceptual (sub)schema :

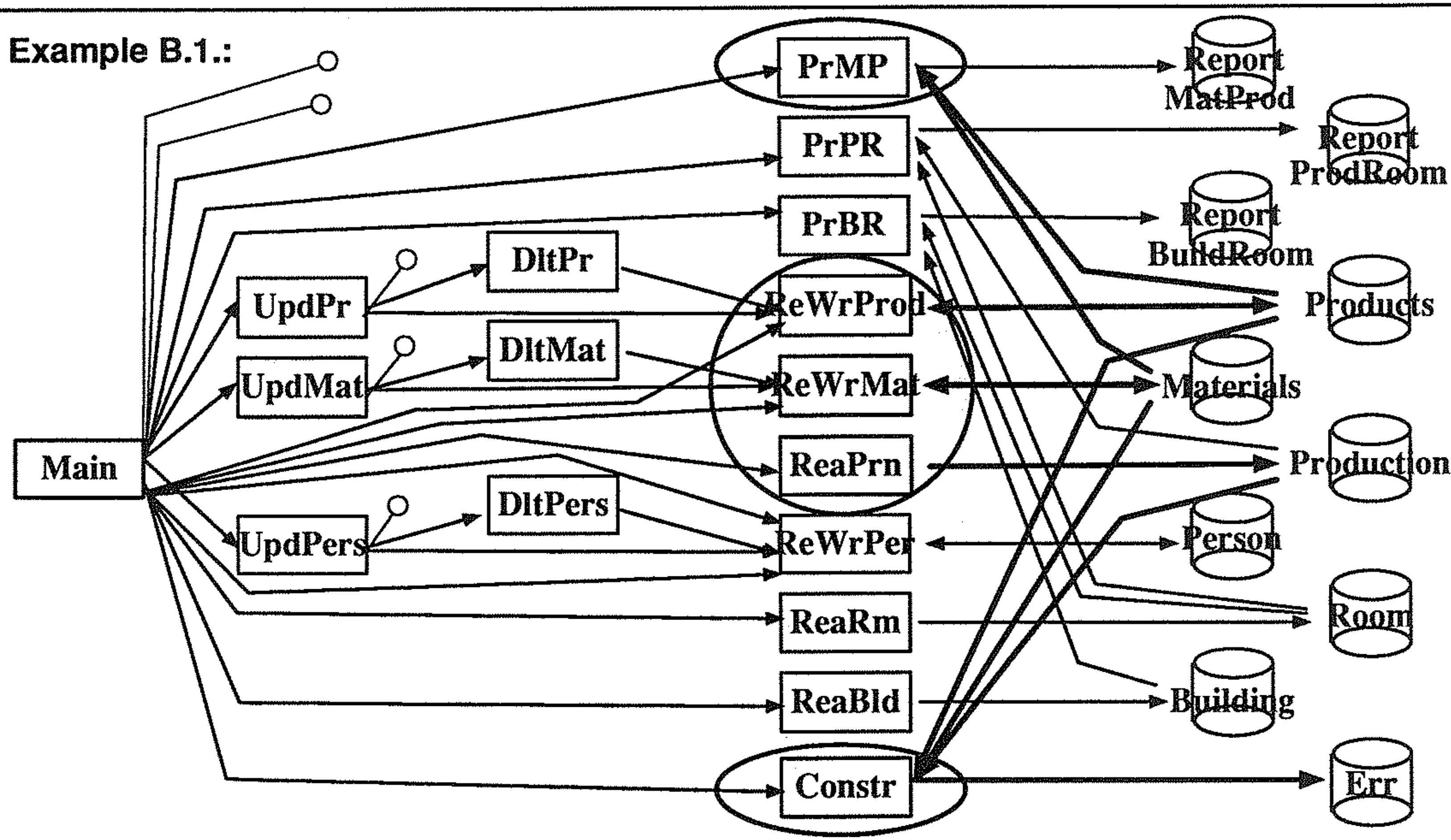
grouping of source files into a worksession

- construct one conceptual schema from the beginning (early integration)

- construct one conceptual schema for each sourcefile (late integration)

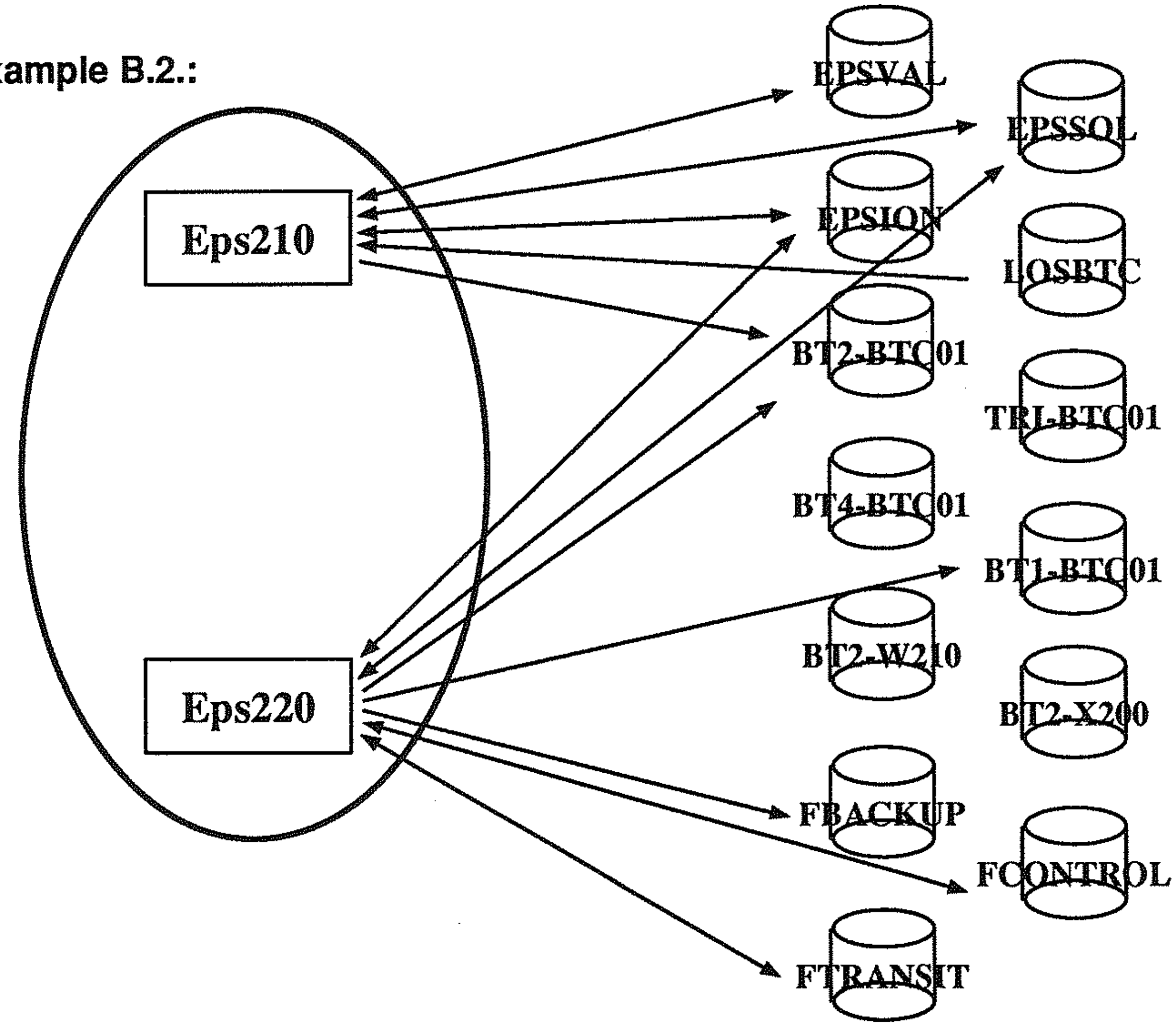
- construct one conceptual schema for sets of source-files (early + late integration)

Example B.1.:



example of a selection of sources to create a partial schema

Example B.2.:


Eps210
Eps220
EPSVAL
EPSSOL
EPSION
LOSBTC
BT2-BTC01
TRI-BTC01
BT4-BTC01
BT1-BTC01
BT2-W210
BT2-X200
FBACKUP
FCONTROL
FTRANSIT

PART 2 : DATA STRUCTURE EXTRACTION

Step a : data structure extraction :

a.1.: DMS part expression

- Finding the logical entity types

- Finding the logical attributes

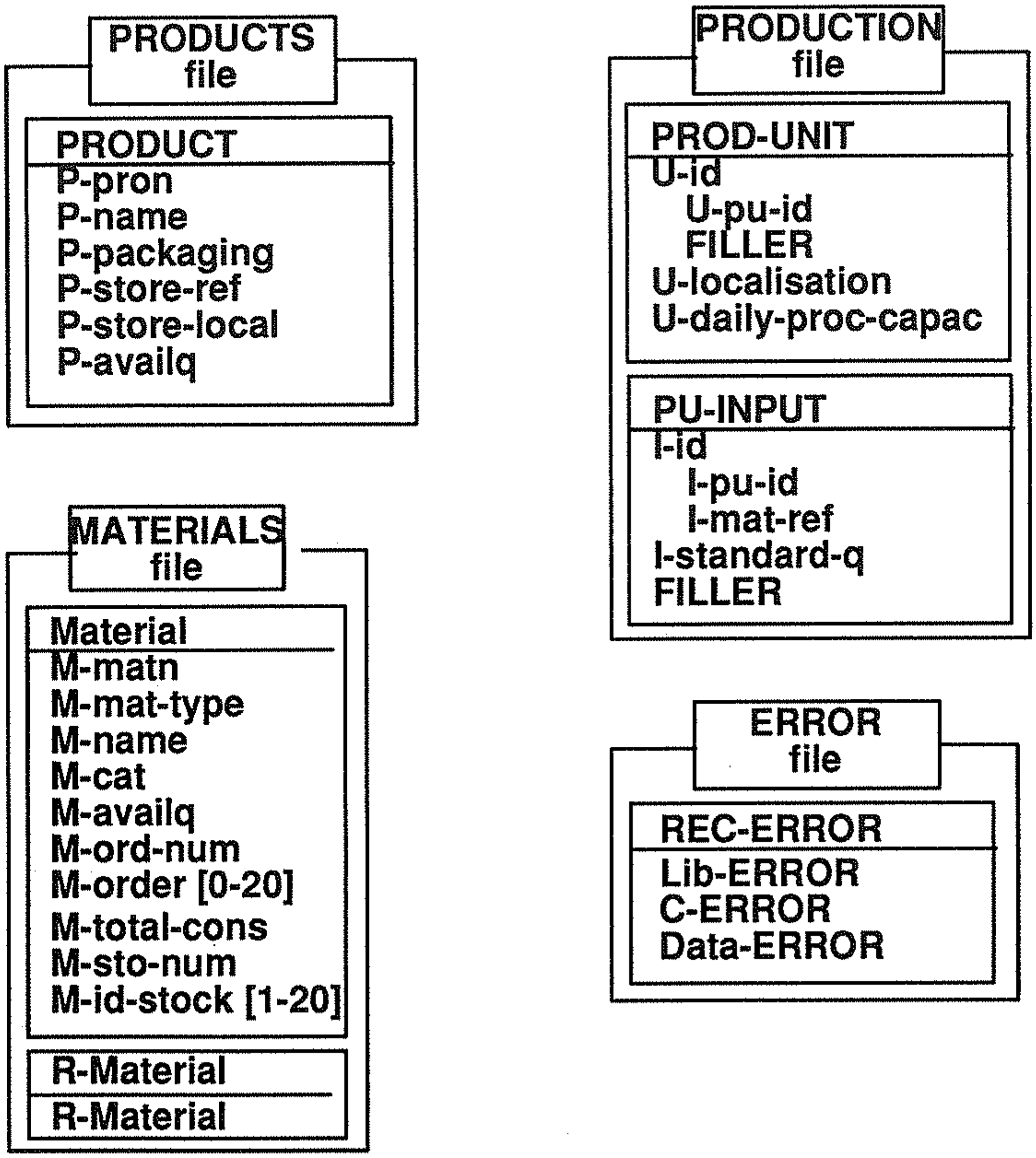
- Finding the logical relationship types

Example A.1.: Extraction of record descriptions from the FILE SECTION

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT PRODUCTION ASSIGN TO DSK:FUP;
      ORGANIZATION IS INDEXED;
      RECORD KEY IS U-id.
   SELECT PRODUCTS ASSIGN TO DSK:FSP;
      ORGANIZATION IS INDEXED;
      RECORD KEY IS P-pron;
      ALTERNATE RECORD KEY IS P-store-ref WITH DUPLICATES.
   SELECT MATERIALS ASSIGN TO DSK:FSM;
      ORGANIZATION IS INDEXED;
      RECORD KEY IS M-matn
      FILE STATUS IS W-STATUS.
   SELECT ERROR-DB ASSIGN TO DSK:FUP.
```

```
DATA DIVISION.
FILE SECTION.
FD PRODUCT; LABEL RECORDS ARE STANDARD.
   01 PRODUCT.
      03 P-pron PIC IS 9(6).
      03 P-name PIC IS X(15).
      03 P-packaging PIC IS X(15).
      03 P-store-ref PIC IS 9(2).
      03 P-store-local PIC IS X(15).
      03 P-availq PIC IS 9(4).
FD PRODUCTION; LABEL RECORDS ARE STANDARD.
   01 PROD-UNIT.
      03 U-id.
         05 U-pu-id PIC IS 9(3).
         05 FILLER PIC IS 9(5); VALUE IS 0.
      03 U-localisation PIC IS X(15).
      03 U-daily-proc-capac PIC IS 9(5).
   01 PU-INPUT.
      03 I-id.
         05 I-pu-id PIC IS 9(3).
         05 I-mat-ref PIC IS 9(5).
      03 I-standard-q PIS IS 9(4).
      03 FILLER PIC IS X(16).
```

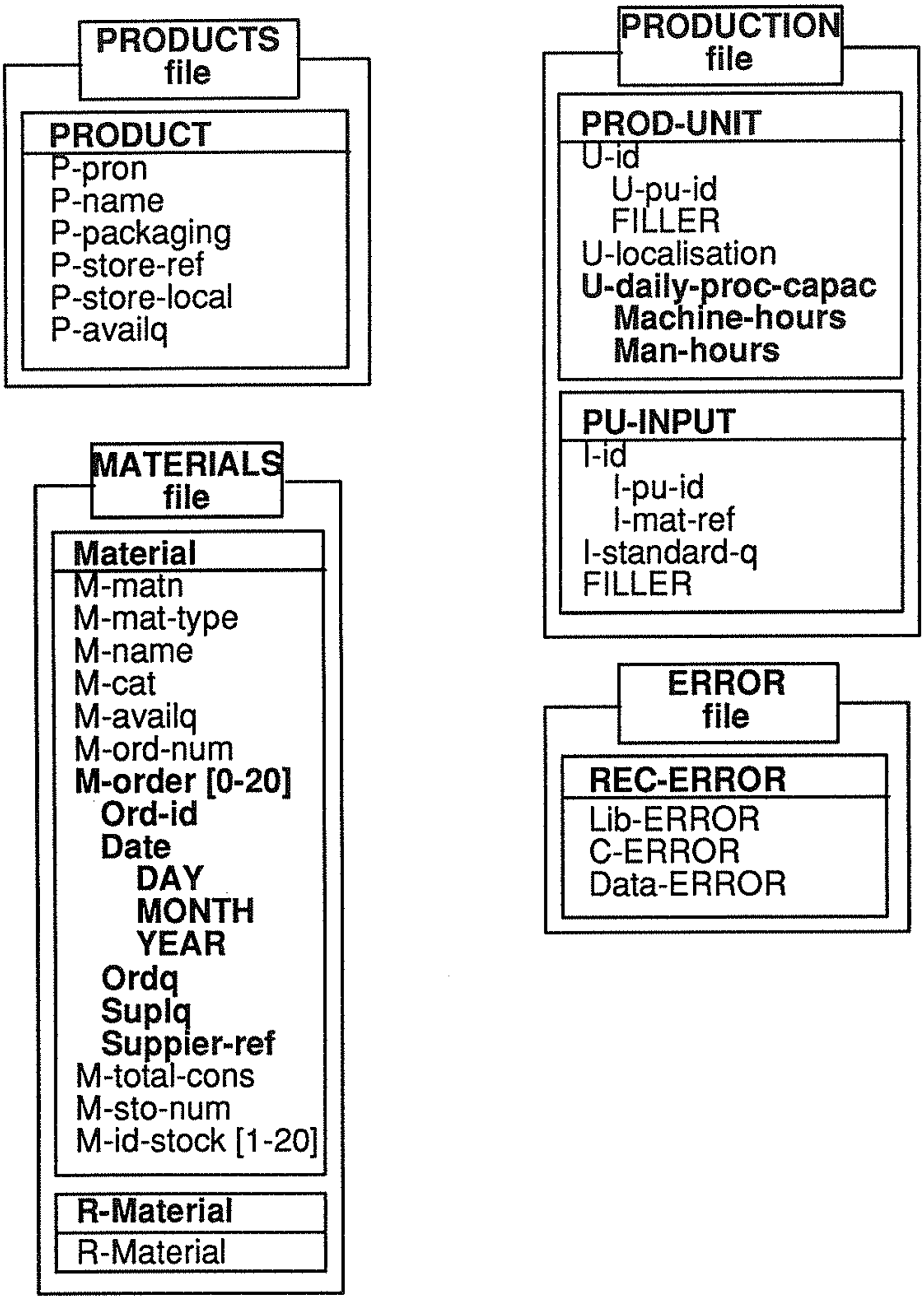
```
FD MATERIALS; LABEL RECORDS ARE STANDARD.
  01 R-MATERIAL PIC IS X(1024).
  01 MATERIAL.
    03 M-matn PIC IS 9(5).
    03 M-mat-type PIC IS X.
      88 M-raw-mat value is 'R'.
      88 M-cons-mat value is 'C'.
    03 M-name PIC IS X(15).
    03 M-cat PIC IS X(15).
    03 M-availq PIC IS 9(4).
    03 M-ord-num PIC IS 9(2).
    03 M-order OCCURS 0 TO 20 DEPENDING ON M-ord-num
                PIC IS X(26).
    03 M-total-cons PIC IS 9(2).
    03 M-sto-num PIC IS 9(2).
    03 M-id-stock OCCURS 1 TO 20 DEPENDING ON M-sto-num
                   PIC IS 9(2).
FD ERROR-DB; LABEL RECORDS ARE STANDARD.
  01 REC-ERROR.
    03 Lib-ERROR PIC IS X(126).
    03 C-ERROR PIC IS 9(2).
    03 Data-ERROR PIC IS X(1024).
```


PRODUCTS file
PRODUCT
P-pron
P-name
P-packaging
P-store-ref
P-store-local
P-availq
MATERIALS file
Material
M-matn
M-mat-type
M-name
M-cat
M-availq
M-ord-num
M-order [0-20]
M-total-cons
M-sto-num
M-id-stock [1-20]
R-Material
R-Material
PRODUCTION file
PROD-UNIT
U-id
U-pu-id
FILLER
U-localisation
U-daily-proc-capac
PU-INPUT
I-id
I-pu-id
I-mat-ref
I-standard-q
FILLER
ERROR file
REC-ERROR
Lib-ERROR
C-ERROR
Data-ERROR

Example A.2.: structural hiding

```
PROCEDURE DIVISION.

....

UPDATE-ORDERS.

    MOVE TODAY-DDMMYY TO Date OF Order.      !!!!!!!

    ....

    MOVE Order TO M-Order (INDX) OF MATERIAL.

    WRITE MATERIAL.

    ....

PROD-CAP.

    ....

    MOVE Production-Capacity TO U-daily-prod-capac.

    .....

    MOVE TODAY-MMDDYY TO Date OF Order.  !!!!!!!

    .....

    MOVE Order TO W-Order.
```

```
WORKING STORAGE SECTION.
    01 TODAY-DDMMYY.
        03 DAY PIC IS 99.
        03 MONTH PIC IS 99.
        03 YEAR PIC IS 99.
    01 TODAY-MMDDYY.
        03 MONTH PIC IS 99.
        03 DAY PIC IS 99.
        03 YEAR PIC IS 99.
    01 Production-Capacity
        03 Machine-hours  PIC IS 99.
        03 Man-hours PIC IS 999.
    01 Order.
        03 Ord-id  PIC IS 9(8).
        03 Date PIC IS X(6).
        03 Ordq PIC IS 9(4).
        03 Suplq PIC IS 9(4).
        03 Supplier-ref PIC IS 9(4).
```


PRODUCTS file
PRODUCT
P-pron
P-name
P-packaging
P-store-ref
P-store-local
P-availq
MATERIALS file
Material
M-matn
M-mat-type
M-name
M-cat
M-availq
M-ord-num
M-order [0-20]
Ord-id
Date
DAY
MONTH
YEAR
Ordq
Suplq
Suppier-ref
M-total-cons
M-sto-num
M-id-stock [1-20]
R-Material
R-Material
PRODUCTION file
PROD-UNIT
U-id
U-pu-id
FILLER
U-localisation
U-daily-proc-capac
Machine-hours
Man-hours
PU-INPUT
I-id
I-pu-id
I-mat-ref
I-standard-q
FILLER
ERROR file
REC-ERROR
Lib-ERROR
C-ERROR
Data-ERROR

Step b : Finding integrity constraints

- identifiers
- domain constraints
- referential constraints
- ....

Example B.1.: identifiers

a. COBOL rules:

any identifier is an access key

if an ET has some access keys at least one is an identifier

the entity types of a file have corresponding access key

==> record key ==> identifier

==> alternate key ==> key

record key U-id	==>	ID(PROD-UNIT)	: U-id.
record key U-id	==>	ID(PU-INPUT)	: I-id.
record key M-matn	==>	ID(MATERIAL)	: M-matn.
record key P-pron	==>	ID(PRODUCT)	: P-pron.

alternate key P-store-ref ==> KEY(PRODUCT): P-store-ref.

b. name conventions + smantic knowledge

prefix / suffix : "id"

attribute Ord-id ==> ID(Material.M-order) = Ord-id.

Example B.2.: domain constraints

a. 88-fields in COBOL

01 **MATERIAL.**

03 **M-matn** PIC IS 9(5).

03 **M-mat-type** PIC IS X.

88 **M-raw-mat** value is 'R'.

88 **M-cons-mat** value is 'C'.

03 **M-name** PIC IS X(15).

03 **M-cat** PIC IS X(15).

03 **M-availq** PIC IS 9(4).

03 **M-ord-num** PIC IS 9(2).

03 **M-order** OCCURS 0 TO 20 DEPENDING ON **M-ord-num**
PIC IS X(26).

03 **M-total-cons** PIC IS 9(2).

03 **M-sto-num** PIC IS 9(2).

03 **M-id-stock** OCCURS 1 TO 20 DEPENDING ON **M-sto-num**
PIC IS 9(2).

==> Material.M-mat-type = {'R', 'C'}

b. if ... else if ... else if ... else if ... else .. statements

```
....

S-TRAIT-MVTS.
   IF IND-MVT OF BTC-DATA = "01" ADD 1 to W-NREC01 ELSE
   IF IND-MVT OF BTC-DATA = "21" ADD 1 to W-NREC21 ELSE
   IF IND-MVT OF BTC-DATA = "02" ADD 1 to W-NREC02 ELSE
   IF IND-MVT OF BTC-DATA = "22" ADD 1 to W-NREC22 ELSE
   IF IND-MVT OF BTC-DATA = "04" ADD 1 to W-NREC04 ELSE
   IF IND-MVT OF BTC-DATA = "24" ADD 1 to W-NREC24 ELSE
                                        PERFORM MVTS-ERROR.

....
```

==> BTC-DATA.IND-MVT =

{"01", "21", "02", "22", "04", "24"}

Example B.3.: Referential constraints :

a. Usage of alternate keys + semantic knowledge

```
SELECT PRODUCTS ASSIGN TO DSK:FSP;
   ORGANIZATION IS INDEXED;
   RECORD KEY IS P-pron;
   ALTERNATE RECORD KEY IS P-store-ref WITH DUPLICATES.
....
```

KEY(PRODUCT) : P-store-ref

~~> PRODUCT.P-store-ref is-in

identifier of STORE

b. Programming techniques ==> use complex search-patterns to find referential constraints:

```
[1]  MOVE <*ref-field> TO <*key-field>
[2]  READ <*file> KEY IS <*key-field>
[3]  TEST <*file-*statusfield>
[4]  COND
       NOT-FOUND :  <*error>
       FOUND :      CONTINUE
```

e.g.:

```
COHER-PU-INPUT.
    ...
    MOVE I.mat-ref OF PU-INPUT TO M-matn OF MATERIAL   [1]
    READ MATERIALS KEY IS M-matn OF MATERIAL           [2]
    IF W-STATUS = YGOOD-RESULT                         [3]
      NEXT SENTENCE                                    [4a]
    ELSE
      PERFORM R-STATUS-ERROR.                          [4b]
    ....

R-STATUS-ERROR.
    IF W-STATUS = YNOTFOUND
      MOVE "Error : Material not found " TO Lib-ERROR
      PERFORM ERR-TASK
    ELSE IF W-STATUS NOT= YGOODRESULT
      MOVE "Status error Materials" TO Lib-ERROR
      MOVE W-STATUS TO C-ERROR
      MOVE R-MATERIAL TO Data-ERROR
      PERFORM ERR-TASK.
```

==> PU-INPUT.I-mat-ref is-in MATERIAL.M-matn

c. Naming conventions + semantical knowledge

```
01 MATERIAL.
   03 M-matn PIC IS 9(5).
   03 M-mat-type PIC IS X.
   03 M-name PIC IS X(15).
   03 M-cat PIC IS X(15).
   03 M-availq PIC IS 9(4).
   03 M-ord-num PIC IS 9(2).
   03 M-order OCCURS 0 TO 20 DEPENDING ON M-ord-num.
      05 Ord-id  PIC IS 9(8).
      05 Date PIC IS X(6).
      05 Ordq PIC IS 9(4).
      05 Suplq PIC IS 9(4).
      05 Supplier-ref PIC IS 9(4).
   03 M-total-cons PIC IS 9(2).
   03 M-sto-num PIC IS 9(2).
   03 M-id-stock OCCURS 1 TO 20 DEPENDING ON M-sto-num
                 PIC IS 9(2).
```

==> MATERIAL.M-id-stock is-in identifier of STOCK.

==> MATERIAL.M-order.Supplier-ref is-in identifier of SUPPLIER.

Intermediary result :

i. (several) detailed record descriptions of the logical datafiles of the sources.

ii. Some identifiers of these record descriptions

iii. Some constraints

- referential
- domain
- ....

==> COBOL-compliant optimized schema + referential constraints

PRODUCTS file

PRODUCT
- P-pron
- P-name
- P-packaging
- P-store-ref
- P-store-local
- P-availq

MATERIALS file

Material
- M-matn
- M-mat-type
- M-name
- M-cat
- M-availq
- M-ord-num
- **M-order [0-20]**
  - **Ord-id**
  - **Date**
    - **DAY**
    - **MONTH**
    - **YEAR**
  - **Ordq**
  - **Suplq**
  - **Suppier-ref**
- M-total-cons
- M-sto-num
- M-id-stock [1-20]

R-Material
- R-Material

PRODUCTION file

PROD-UNIT
- U-id
  - U-pu-id
  - FILLER
- U-localisation
- **U-daily-proc-capac**
  - **Machine-hours**
  - **Man-hours**

PU-INPUT
- I-id
  - I-pu-id
  - I-mat-ref
- I-standard-q
- FILLER

ERROR file

REC-ERROR
- Lib-ERROR
- C-ERROR
- Data-ERROR

PROD-UNIT.U-id.FILLER = 0
MATERIAL.M-mat-type = {'R', 'C'}
KEY(PRODUCT) : P-store-ref
id(MATERIAL.M-order) : Ord-id
PRODUCT.P-store-ref is-in id(STORE)
PU-INPUT.I-mat-ref is-in id(MATERIAL)
MATERIAL.M-id-stock is-in id(STOCK)
MATERIAL.M-order.Supplier-ref is-in id(SUPPLIER)

IMPORTANT REMARK:

Within a real RE-session the data extraction and data conceptualisation processes are not really considered in this strict order.

A mixture of Conceptualisation processes and extraction processes are considered.

E.g.:

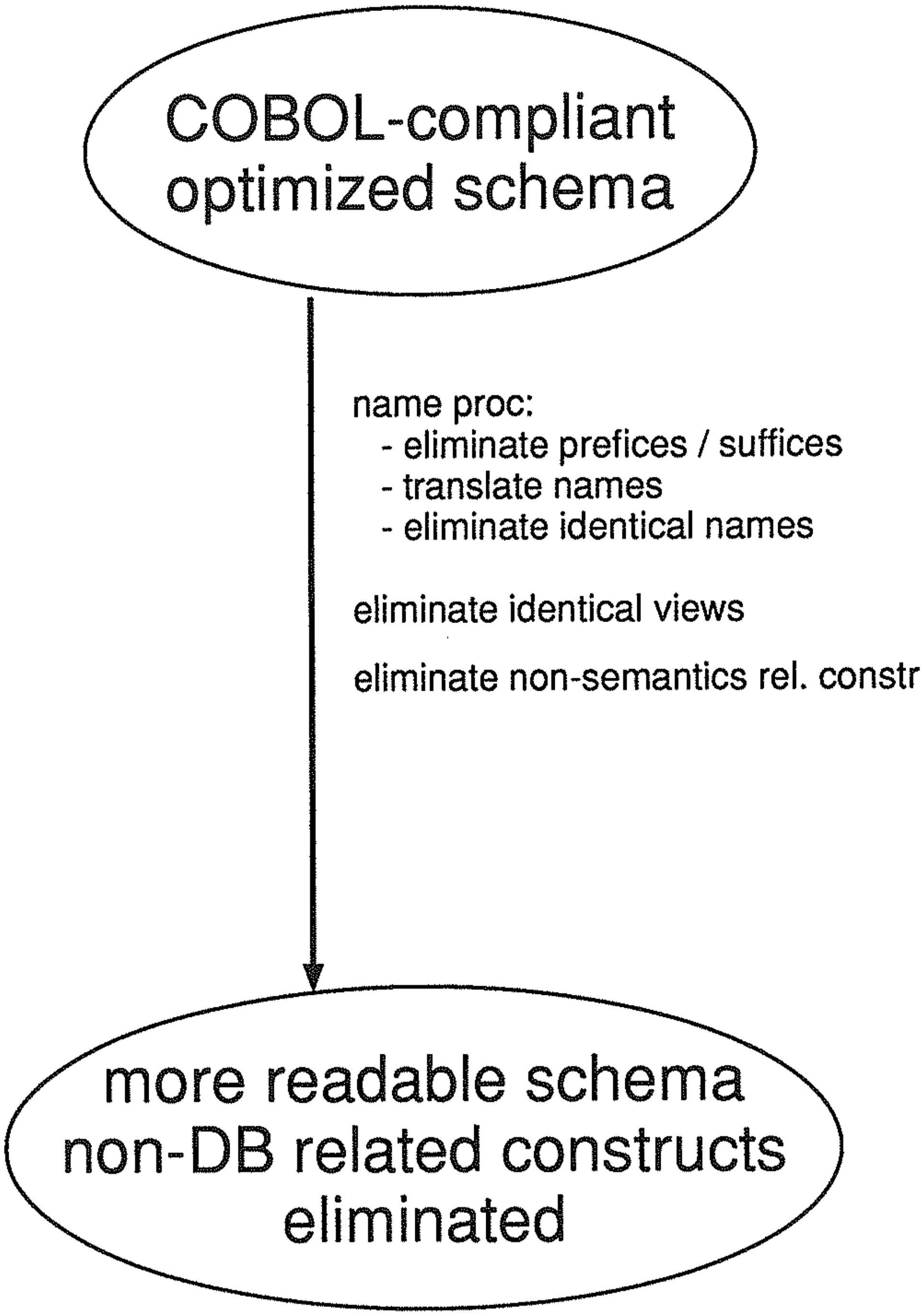
the retrieval of referential constraints may be retarded,

the de-optimisation of a physical /logical description may be considered earlier, ...

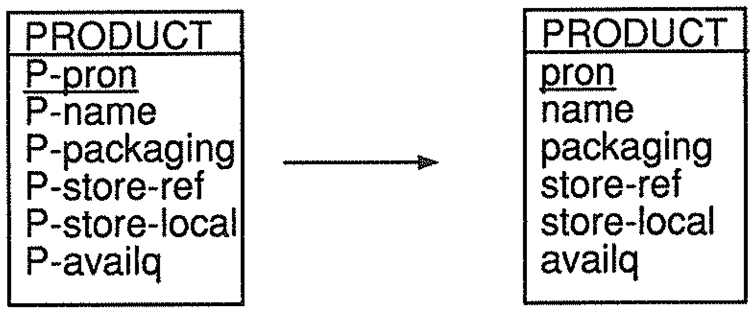
some extraction processes are triggered by a conceptualization process

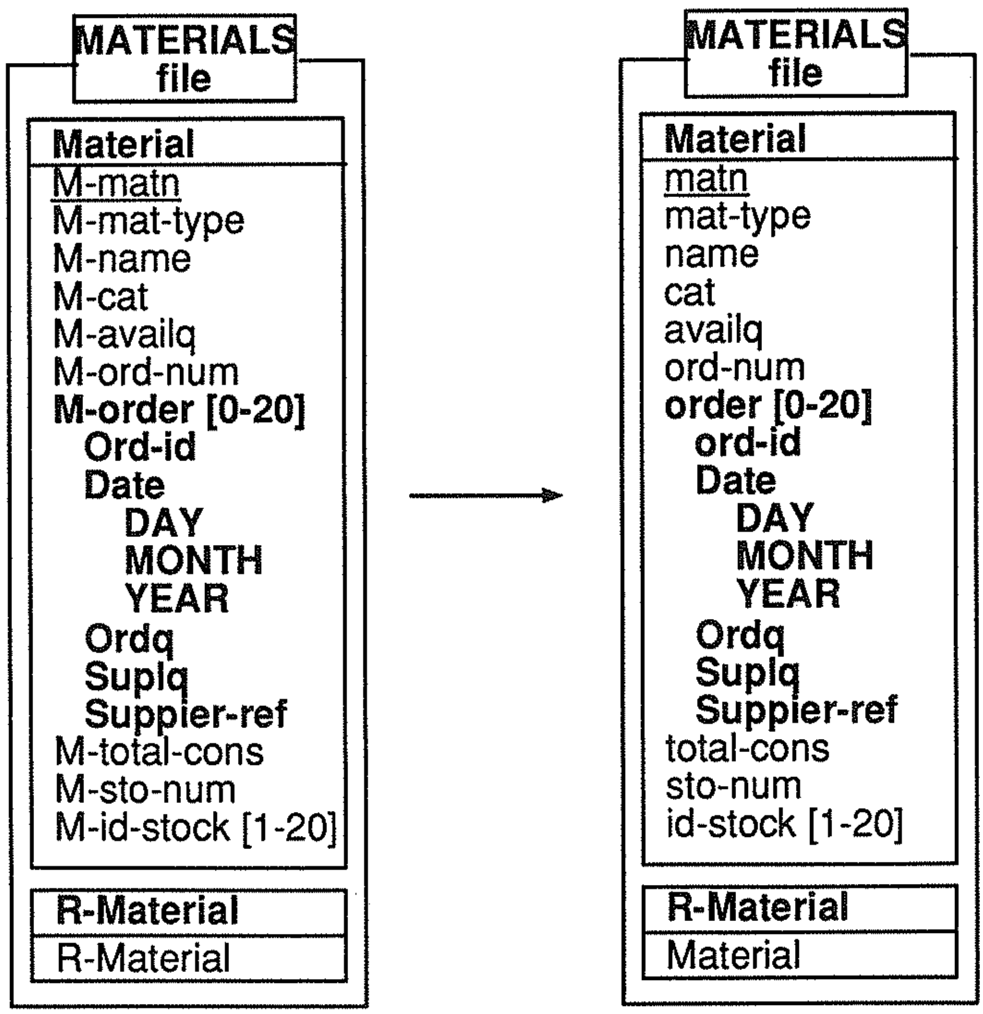
PART 3 : DATA CONCEPTUALISATION

Step a.: schema preprocessing



Example A.1.: prefix removing





Example A.2.:
remove FILLER-fields used for alignment purposes

ART-PHOTO-PH	
DATE-FICHIER-PH	PIC X(8)
FILLER-1	PIC X
TEMPS-FICHIER-PH	PIC X(8)
FILLER-2	PIC X
NOM-MAJ-PH	PIC X(6)
FILLER-3	PIC X
TYPE-MAJ-PH	PIC 9
FILLER-4	PIC X(36)

→

ART-PHOTO-PH	
DATE-FICHIER	PIC X(8)
TEMPS-FICHIER	PIC X(8)
NOM-MAJ	PIC X(6)
TYPE-MAJ	PIC 9

FILLER-1, FILLER-2, FILLER-3 :
used to separate fields

FILLER-4 :
to give the record description the same length as the other record descriptions

Example A.3.:

solve problem of multi-descriptions

remove / integrate redundant descriptions.

```
FD MATERIALS; LABEL RECORDS ARE STANDARD.
   01 R-MATERIAL PIC IS X(1024).
   01 MATERIAL.
      03 M-matn PIC IS 9(5).
      03 M-mat-type PIC IS X.
         88 M-raw-mat value is 'R'.
         88 M-cons-mat value is 'C'.
      03 M-name PIC IS X(15).
      03 M-cat PIC IS X(15).
      03 M-availq PIC IS 9(4).
      03 M-ord-num PIC IS 9(2).
      03 M-order OCCURS 0 TO 20 DEPENDING ON M-ord-num
                    PIC IS X(26).
      03 M-total-cons PIC IS 9(2).
      03 M-sto-num PIC IS 9(2).
      03 M-id-stock OCCURS 1 TO 20 DEPENDING ON M-sto-num
                       PIC IS 9(2).
```

```
....
R-READ-MATERIAL.
  IF KEY-MATERIAL = 0
     START MATERIALS KEY IS > FIRST-MATERIAL
     READ MATERIALS NEXT INTO R-MATERIAL
  ELSE IF KEY-MATERIAL = 1
     READ MATERIALS KEY IS M-matn INTO R-MATERIAL
  ELSE IF KEY-MATERIAL = 2
     READ MATERIALS NEXT INTO R-MATERIAL.
....
```

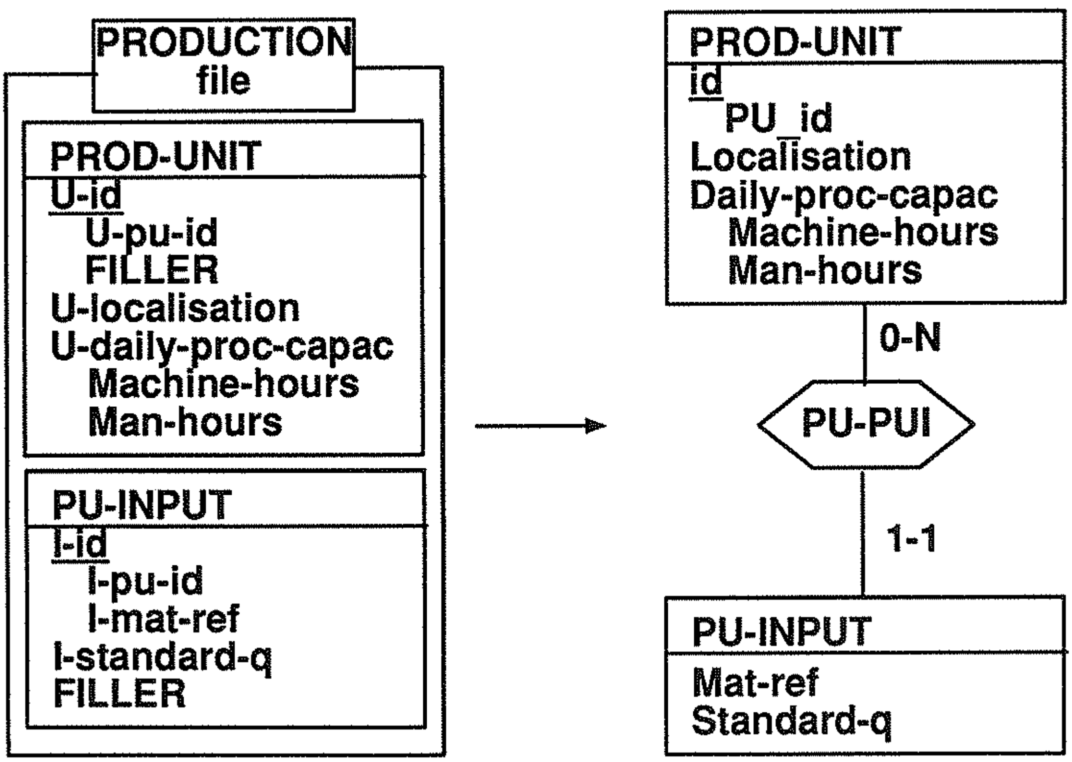
==> R-MATERIAL is only used to READ / WRITE data to the MATERIALS file

==> technical construct of the programmer

Step B : DMS-specific tasks

Example B.1. : de-optimization

- space optimization
  (independent but similar record types in the same file)
- response time optimization
  (e.g.: reduce number of used datafiles)
- ...



id(PROD-UNIT, PU-UNIT) : ...-id

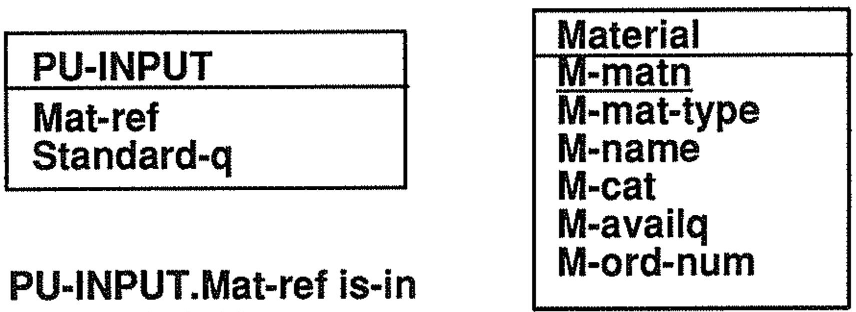
id(PU-INPUT) = Mat-ref, PROD-UNIT

Example B.2.: untranslation

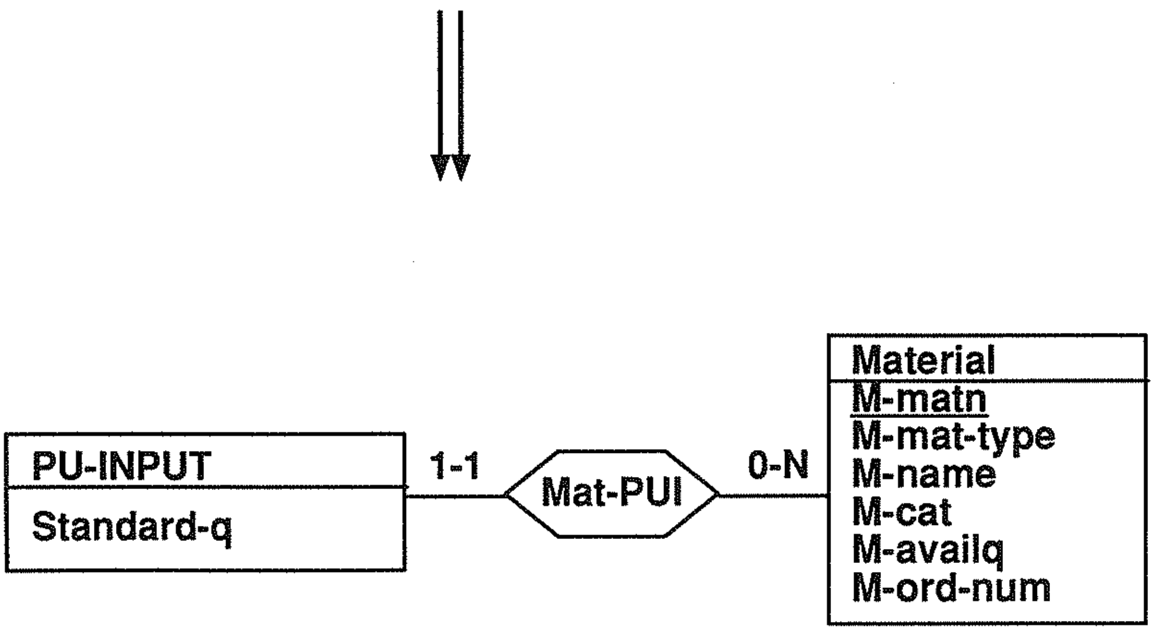
==> execution of transformations in reverse order

e.g.: - referential attribute

==> relationship type



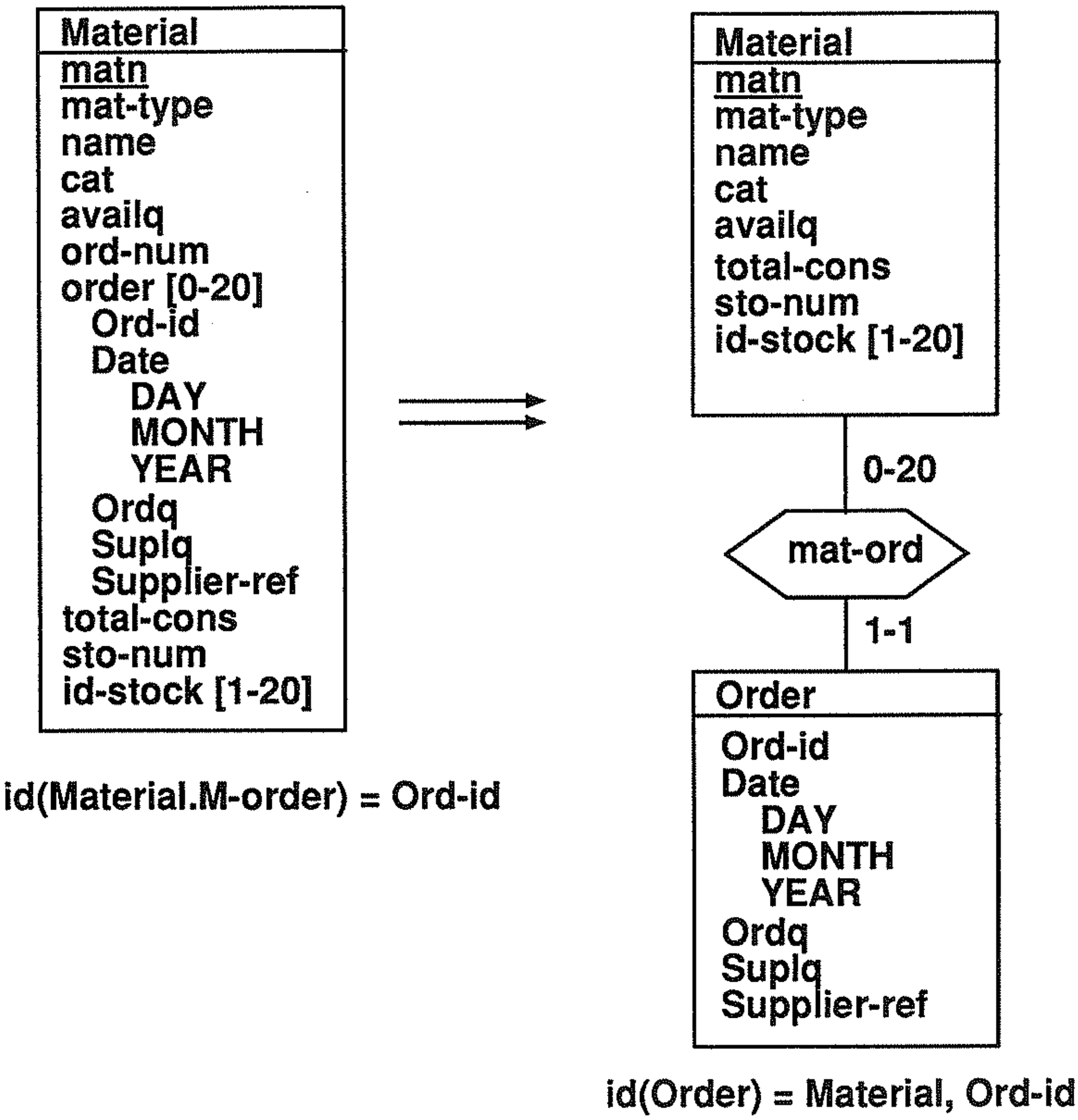
PU-INPUT.Mat-ref is-in
MATERIAL.Matn



Step C : DMS-independent de-optimization

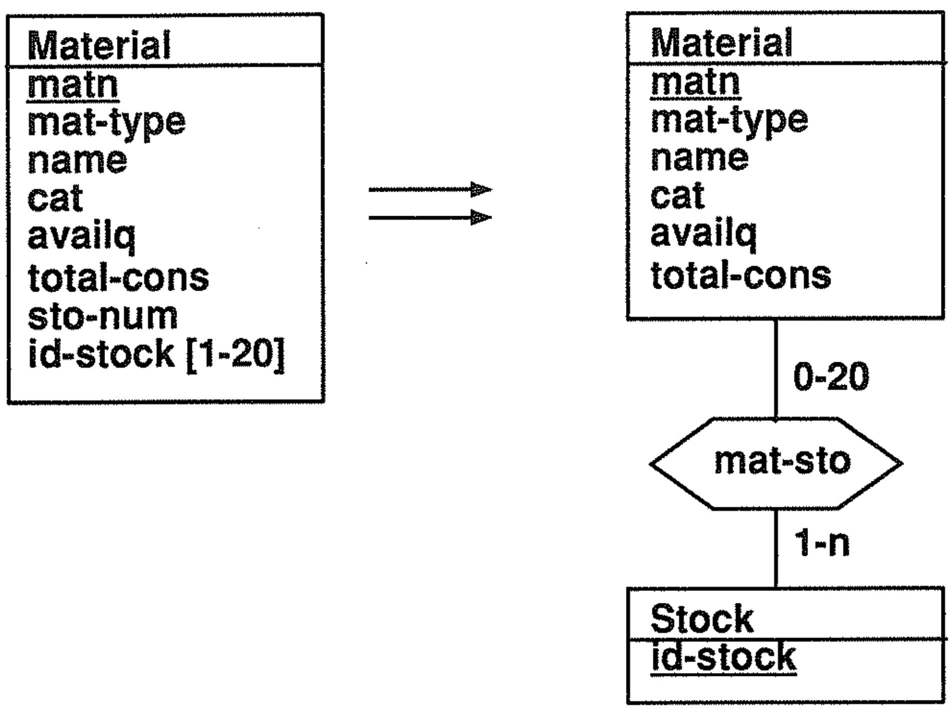
Example C.1.: restructuring

transformation
compound attribute -> entity type



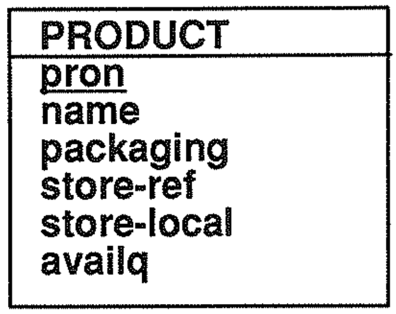
Example C.2.: restructuring

transformation
elementary attribute => entity type

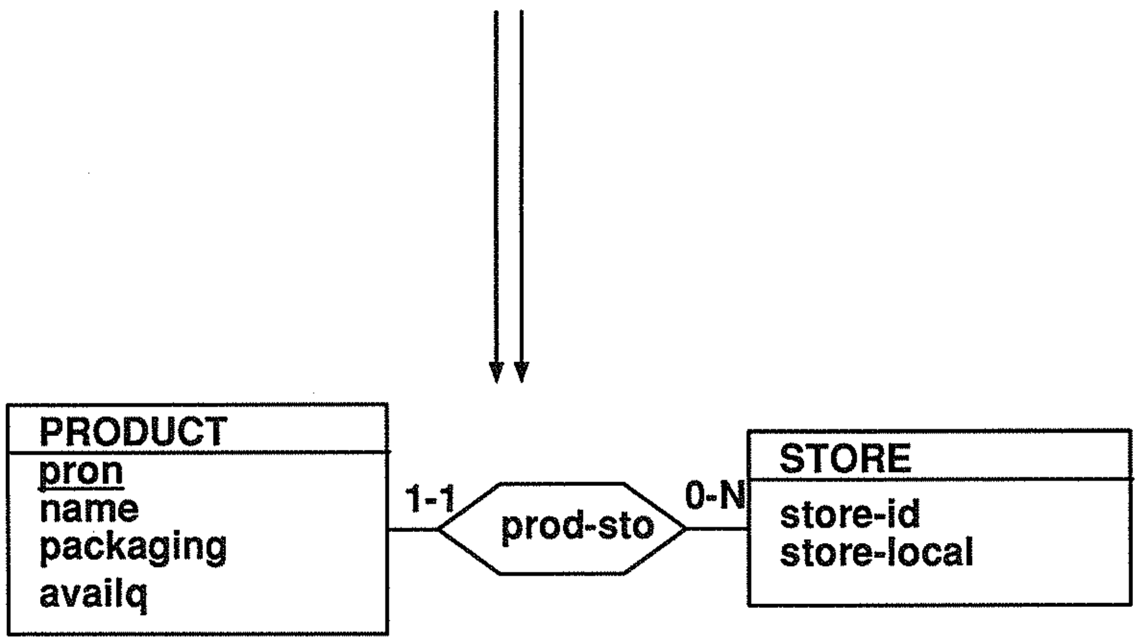


id(stock) : stock.id-stock

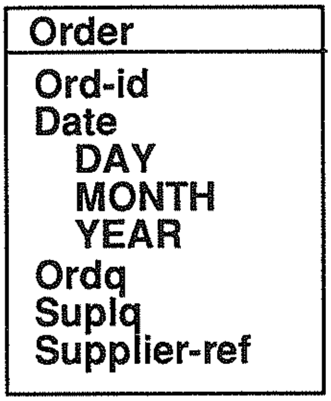
Example C.3.: normalization



referential attribute store-ref.
functional dependency
PRODUCT.store-ref --> PRODUCT.store-local



Example C.4.: normalization: use of domain knowledge



id(Order) : Material, Ord-id

semantics:

a. functional dependency

Ord-id ==> Date, Supplier

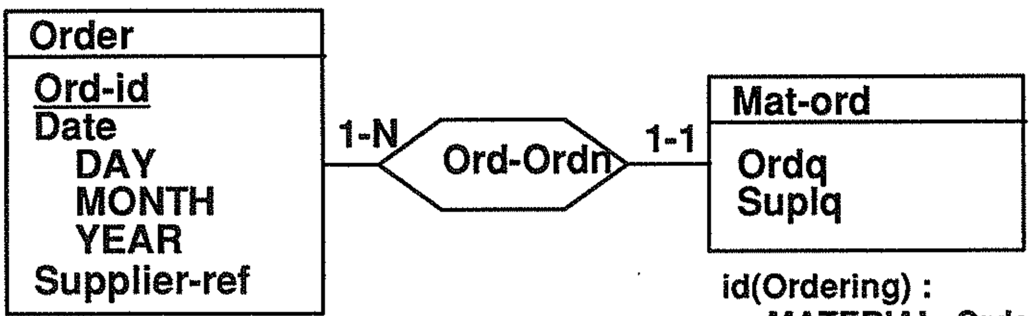
b. We will rename the Order entity type into Material-Order

⇊

action :

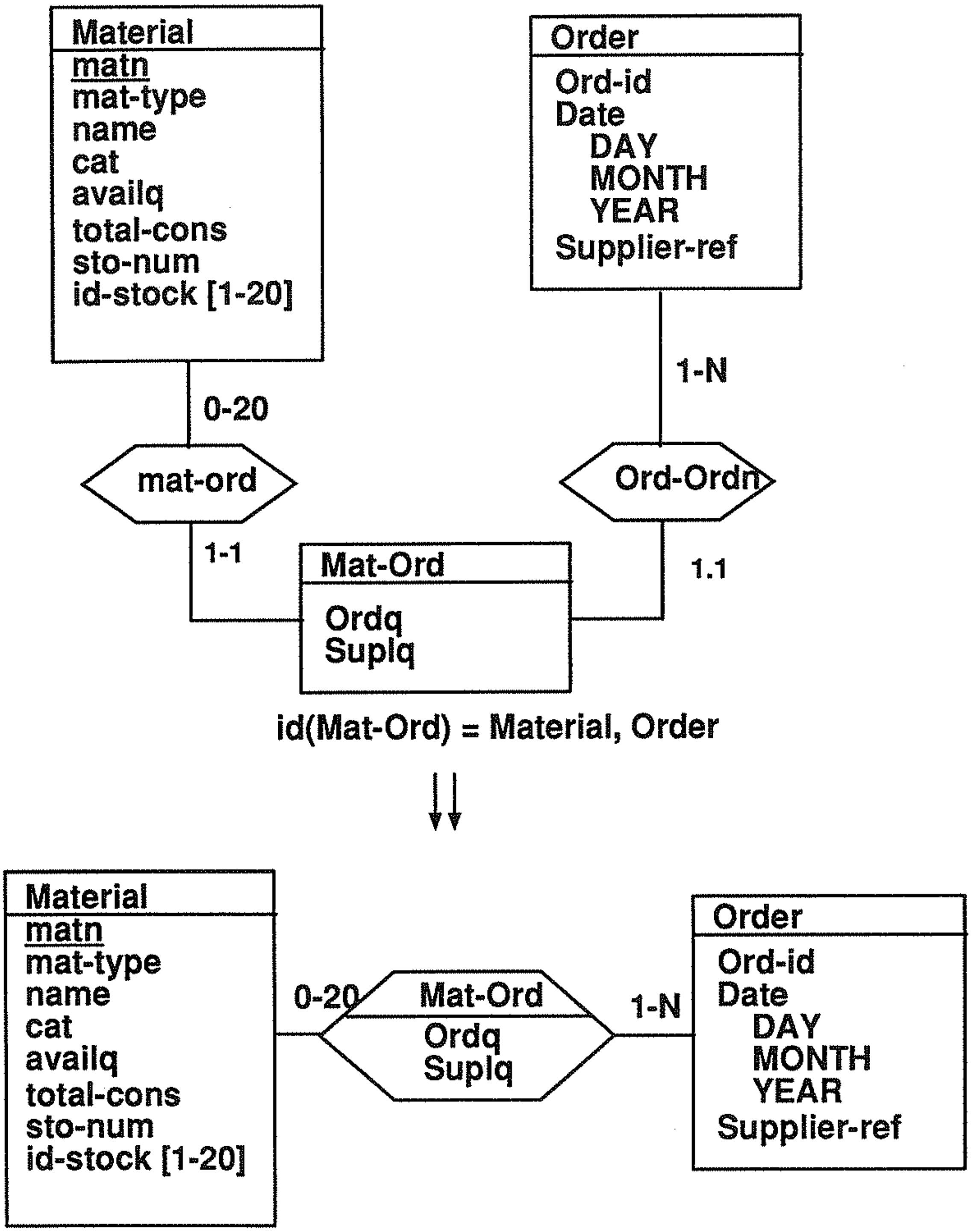
rename Order entity type into Mat-Order
create new entity type Order
with attribute Ordq and Suplq

⇊



id(Ordering) :
MATERIAL, Order

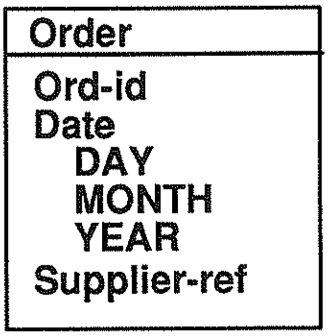
Example C.5.: conceptual restructuring



Example C.6 : restructuring:

creation of a new entity type

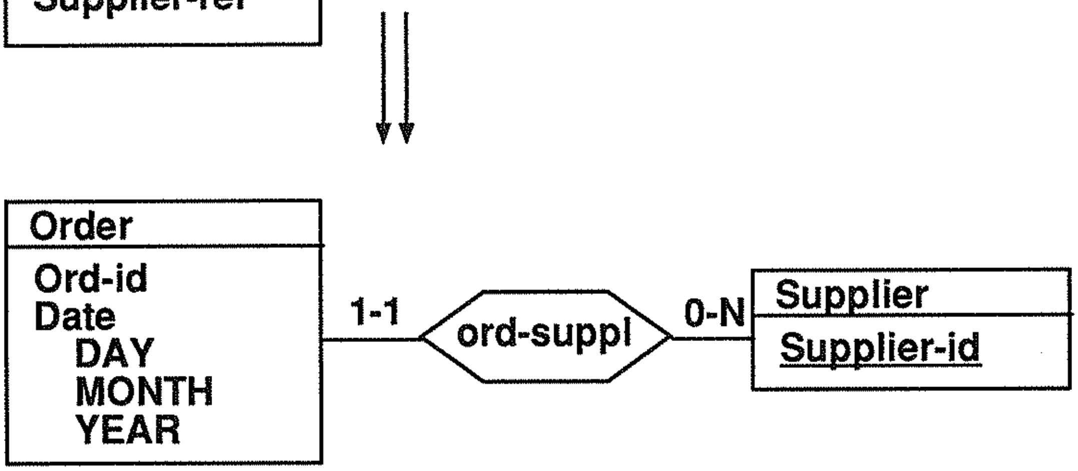
+ transformation of the ref-att into a relationship type



semantic knowledge :

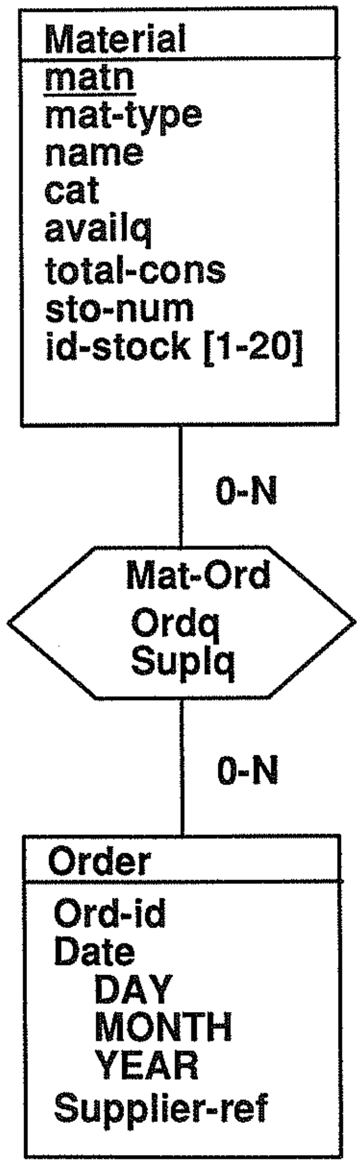
Supplier-ref: referential attribute

==> create ET Supplier



Example C.7 : conceptual restructuring

creation of gen/spec relationship



mat-type = {'R', 'C'}

source-code:

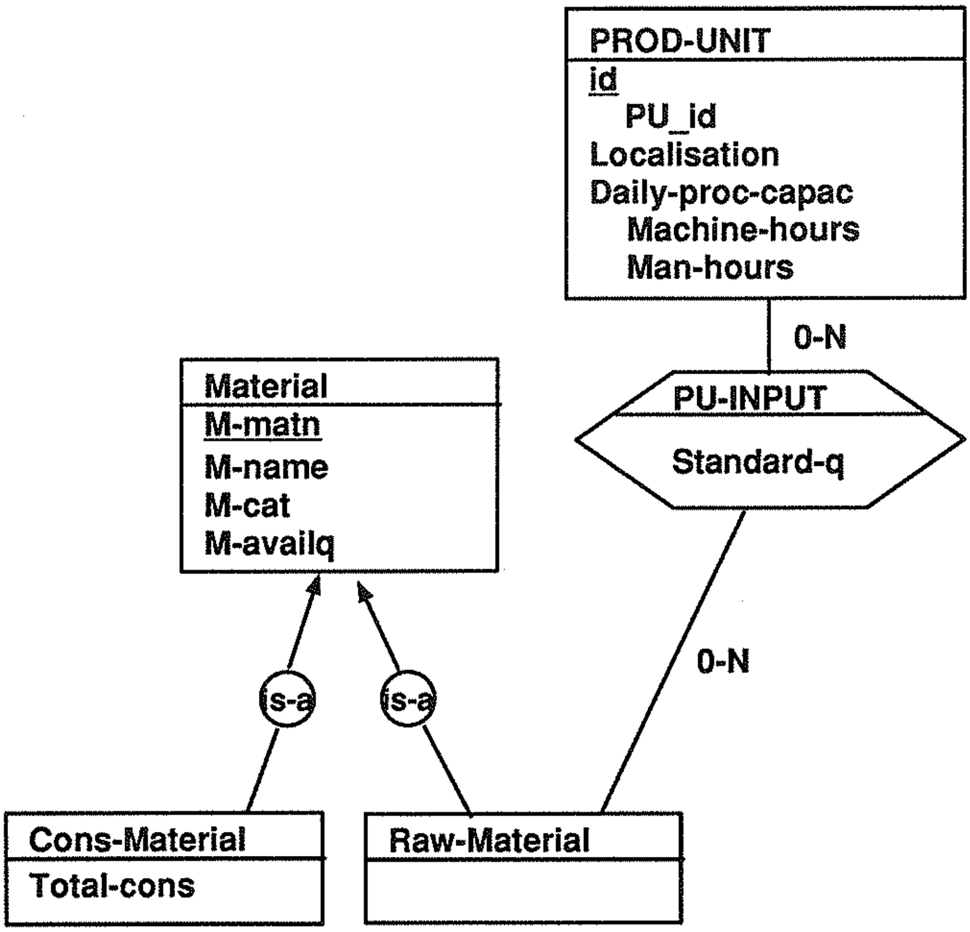
```
S-update-MATERIAL.
  IF W-CONS-MAT
     PERFORM R-READ-MATERIAL
     PERFORM RECALCULATE-TOTAL-CONS
  ELSE IF W-RAW-MAT
     PERFORM R-READ-MATERIAL
     PERFORM R-READ-PRODUCTION
     IF M-AVAILQ OF MATERIAL <P-STANDARD-Q OF PROD-UNIT
        PERFORM ORDER-MAT.
  ....
CALCULATE-TOTAL-CONS.
  IF W-CONS-MAT
     ADD W-INCREM-CONS TO M-TOTAL-CONS OF MATERIAL
     REWRITE MATERIAL.
  ....
```

consequence:

1. **M-TOTAL-CONS is only used if a CONS-MATERIAL is considered**

2. **if a RAW-MATERIAL is considered there is a participation in PU-INPUT.**

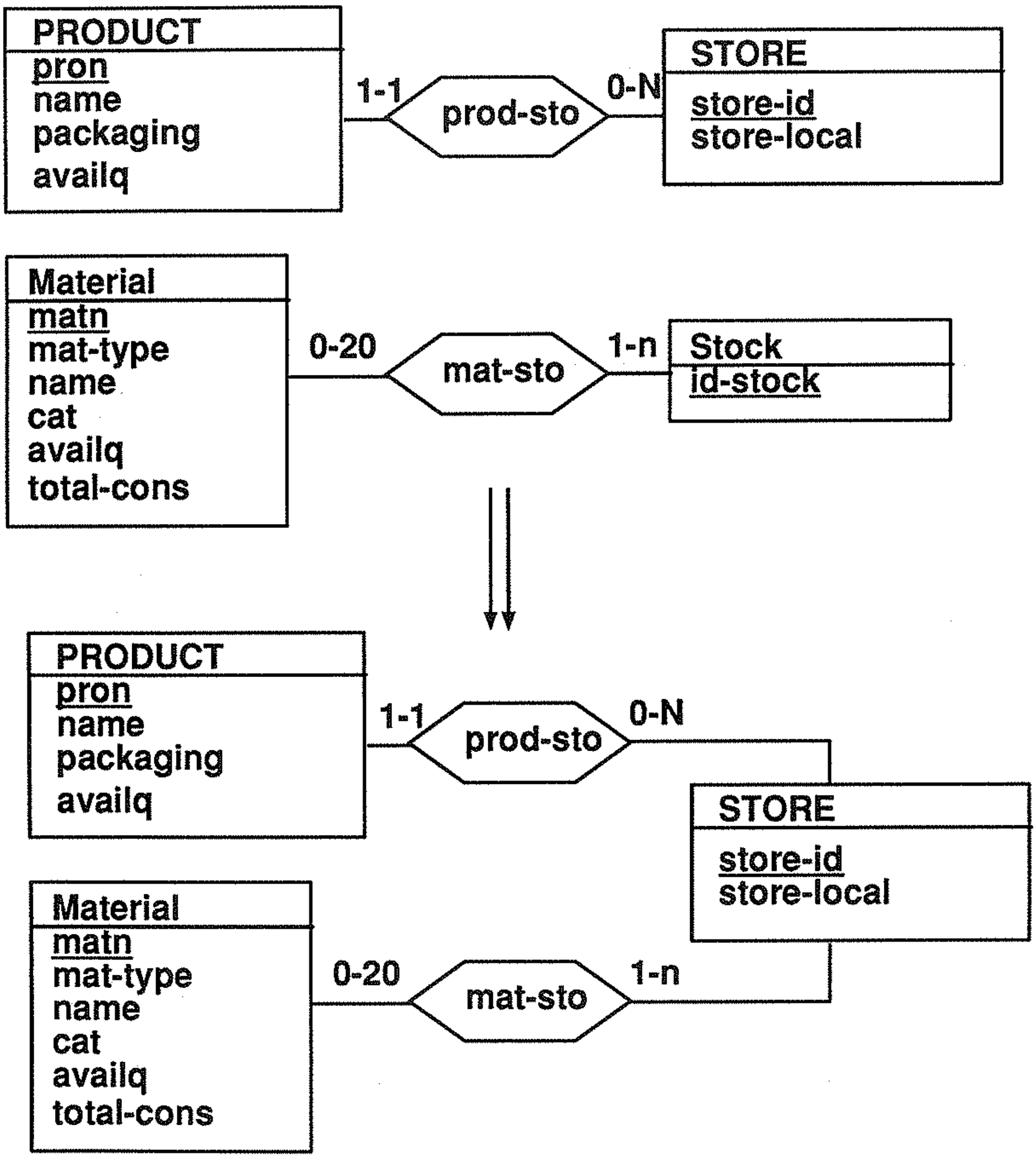
==> supertype / subtype creation


PROD-UNIT
id
PU_id
Localisation
Daily-proc-capac
Machine-hours
Man-hours
0-N
PU-INPUT
Standard-q
0-N
Material
M-matn
M-name
M-cat
M-availq
is-a
is-a
Cons-Material
Total-cons
Raw-Material

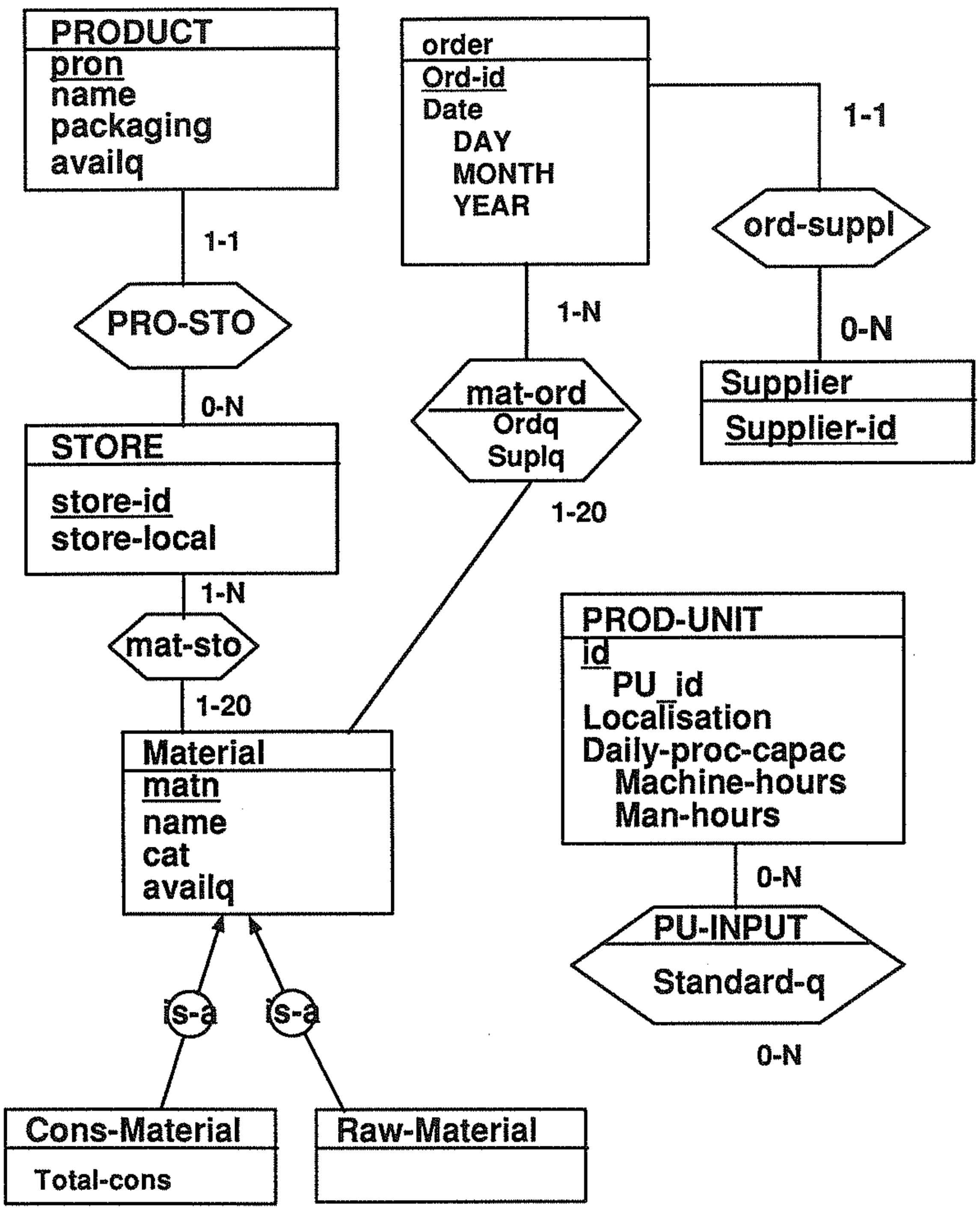
Example C.8.: conceptual restructuring

STORE and STOCK are synonyms of the same ET

==> integration within a schema



RESULT :


PRODUCT
pron
name
packaging
availq
1-1
PRO-STO
0-N
STORE
store-id
store-local
1-N
mat-sto
1-20
Material
matn
name
cat
availq
is-a
is-a
Cons-Material
Total-cons
Raw-Material
order
Ord-id
Date
DAY
MONTH
YEAR
1-N
mat-ord
Ordq
Suplq
1-20
1-1
ord-suppl
0-N
Supplier
Supplier-id
PROD-UNIT
id
PU_id
Localisation
Daily-proc-capac
Machine-hours
Man-hours
0-N
PU-INPUT
Standard-q
0-N

Reverse Engineering Concepts

(Methodological Guide : Vol I : Chapter 2)

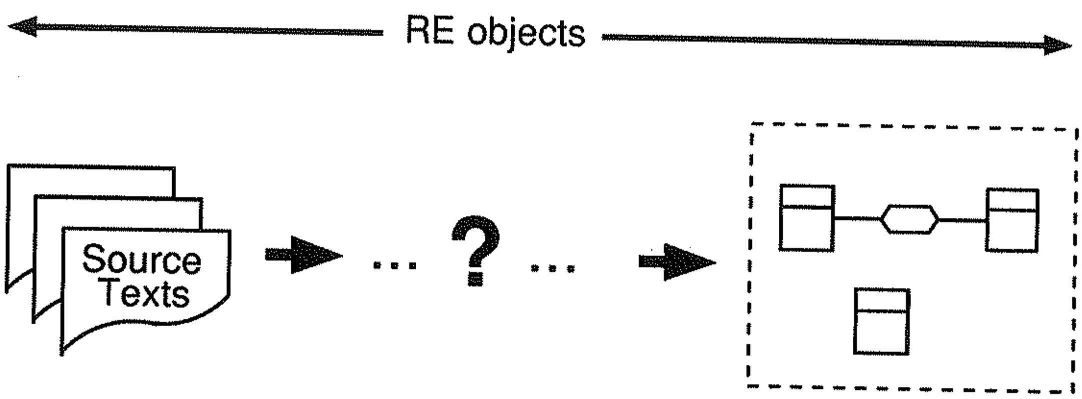
Contents

1. Principles of the modelization 3
2. Required characteristics of the model 7
3. The main concepts 8
4. Typing concepts 11
5. Constraints 13
6. Access structures 15
7. Physical Structures 17
8. Statements 19

1. Principles of the modelization

Goal : to propose concepts which modelize the RE objects, i.e. the data specifications.

What?



How? two questions :

- How many models?
- Which one(s)?

1st Observation :

The concepts managed by diverse DBMSs are generalizable (cf. forward engineering).

Examples :

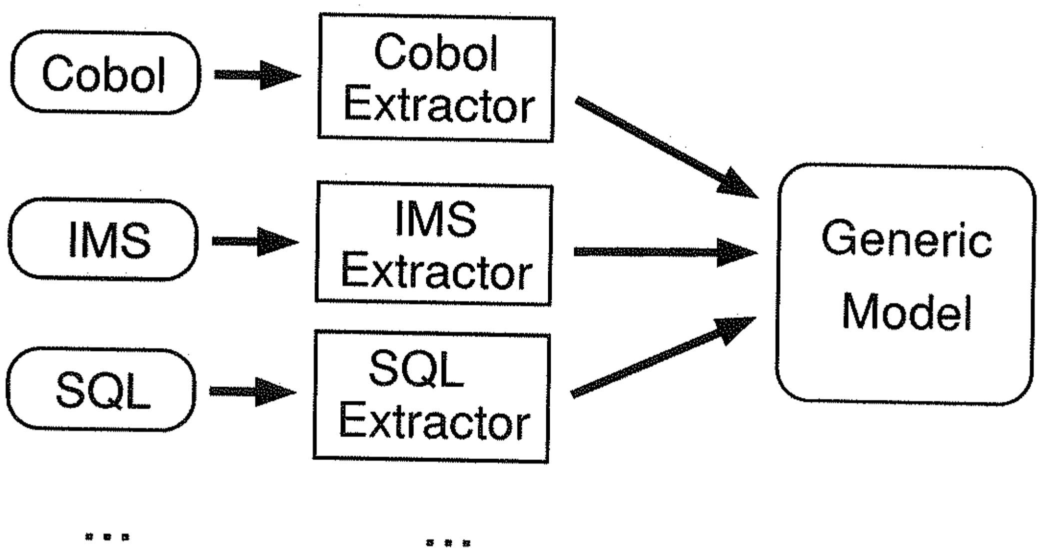
Cobol	**Relational**	**IMS**	**Codasyl**
record type	table	segment type	record type
field	column	field	data-item
record key	index	sequence field	calc-key
pic x(10)	char(10)	bytes = 10	character 10
file	schema	database	area

Conclusion : There could be a single model to record the data specifications resulting from an immediate analysis of the source texts, whatever the DBMS.

Benefits : - genericity, reusability (numerous DBMS).

Drawbacks : - abstract
- simplification

Consequence for a tool :



2nd Observation :

The various levels of data abstraction are generalizable.

Example :

Physical	MAG (logical)	Entity/ Relationship	NIAM
Cobol record type	article type	entity type	no-lot
Cobol field	item	attribute	lot
Codasyl set type	access path	relationship type	bridge type

Conclusion : Only one model during the whole RE life cycle of the data specifications.

Benefits :
- flexibility of processes triggering.
- no intermediary models which constrains to mandatory steps, translations rules, etc.
- genericity for RE processes, too.

Drawbacks :
- abstract
- methodology must be defined somewhere else

2. Required characteristics of the model

Semantical Richness

The model should include the most recent concepts which allows to describe better the semantical complexity of the real world.

Completeness

The model should include all the needed concepts for database reverse engineering (more than usual DB-forward CASE tool dictionaries).

Minimality

The model should not include two concepts which covers the same reality.

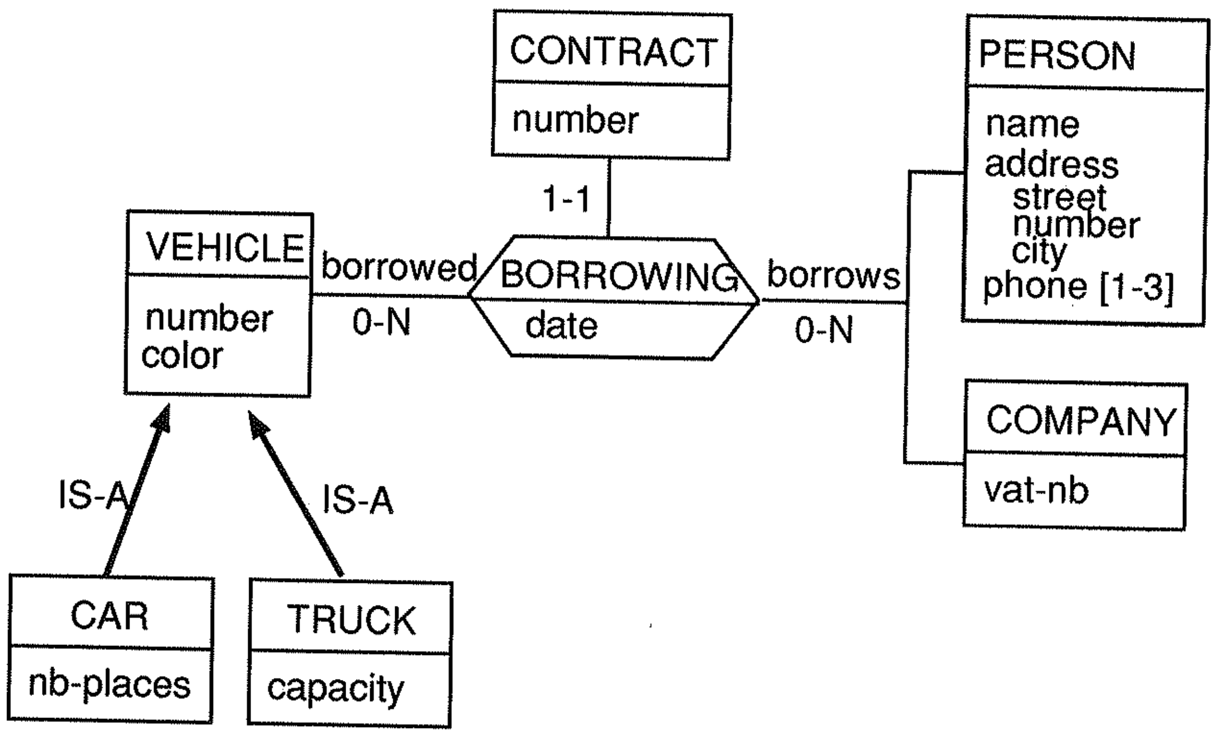
Simplicity

The model should not include useless concepts. (increasing the number of concepts in a model means increasing its complexity).

3. The main concepts

- **Entity type** : name
- **Relationship type** : name
  roles : name, cardinalities
- **Attribute** : name, repetitivities, start position
- **Generalization/Specialization** of entity types

Example 1 :



Example 2 (COBOL) :

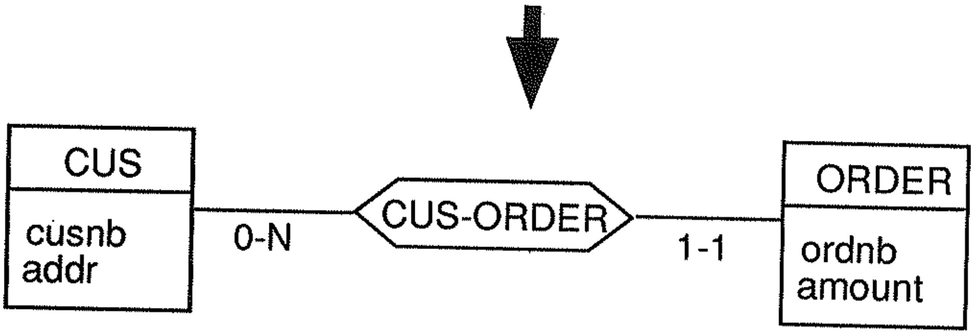
```
01       A-ENRVS30-W.
  03     A0A-ECRVS30A.
    05   AOA-CDBLC          PIC XXXX.
    05   AOA-CDSGE          PIC XX.
    05   AOA-NOORD.
      07 AOA-CRNUM001       PIC X.
      07 AOA-CRNUM002       PIC X.
      07 AOA-CRNUM003       PIC XXX.
    05   A0A-NOLOT          PIC 99.
    05   AOA-CDCTE          PIC X.
    05   AOA-CDTYPMVT       PIC X.
    05   AOA-DTECREDOC.
      07 AOA-DTJOU          PIC XX.
      07 AOA-DTMOI          PIC 99.
      07 AOA-DTANN          PIC 99.
    05   AOA-EXPMCH.
      07 AOA-DTJOU          PIC XX.
      07 AOA-DTMOI          PIC 99.
      07 AOA-DTANN          PIC 99.
    05   AOA-CDIND          PIC X  OCCURS 12.
    05   AOA-CDPAI          PIC XX.
    05   AOA-CD-PAI-R       REDEFINES AOA-CDPAI
                            PIC 99.
    05   FILLER             PIC X(39).
  03     AOA-ECRVS30A-1.
    05   AOA-CDBLC-1        PIC XXXX.
    05   AOA-CDSGE-1        PIC XX.
    05   AOA-NOORD-1.
      07 AOA-CRNUM001-1     PIC X.
      07 AOA-CRNUM002-1     PIC X.
      07 AOA-CRNUM003-1     PIC XXX.
    05   A0A-NOLOT-1        PIC XX.
    05   AOA-CDCTE-1        PIC X.
    05   AOA-CDTYPMVT-1     PIC X.
    05   AOA-DTECREDOC-1.
      07 AOA-DTJOU-1        PIC XX.
      07 AOA-DTMOI-1        PIC XX.
      07 AOA-DTANN-1        PIC XX.
    05   AOA-EXPMCH-1.
      07 AOA-DTJOU-1        PIC XX.
      07 AOA-DTMOI-1        PIC XX.
      07 AOA-DTANN-1        PIC XX.
    05   AOA-CDIND-1        PIC X  OCCURS 12.
    05   AOA-CDPAI-1        PIC XX.
    05   FILLER             PIC X(39).
```

→

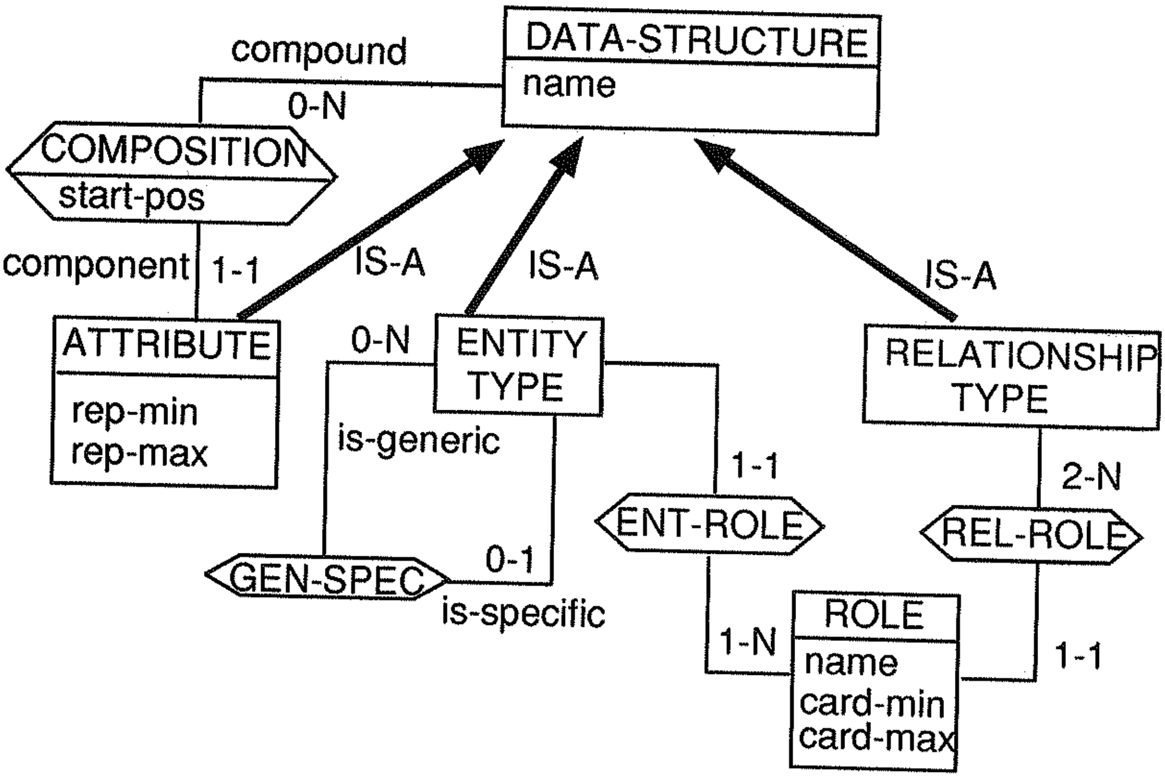
```
A-ENRVS30A-W
A0A-ECRVS30A
    AOA-CDBLC
    AOA-CDSGE
    AOA-NOORD
        AOA-CRNUM001
        AOA-CRNUM002
        AOA-CRNUM003
    A0A-NOLOT
    AOA-CDCTE
    AOA-CDTYPMVT
    AOA-DTECREDOC
        AOA-DTJOU
        AOA-DTMOI
        AOA-DTANN
    AOA-EXPMCH
        AOA-DTJOU
        AOA-DTMOI
        AOA-DTANN
    AOA-CDIND [1-12]
    AOA-CDPAI
    AOA-CD-PAI-R
    FILLER
AOA-ECRVS30A-1
    AOA-CDBLC-1
    AOA-CDSGE-1
    AOA-NOORD-1
        AOA-CRNUM001-1
        AOA-CRNUM002-1
        AOA-CRNUM003-1
    A0A-NOLOT-1
    AOA-CDCTE-1
    AOA-CDTYPMVT-1
    AOA-DTECREDOC-1
        AOA-DTJOU-1
        AOA-DTMOI-1
        AOA-DTANN-1
    AOA-EXPMCH-1
        AOA-DTJOU-1
        AOA-DTMOI-1
        AOA-DTANN-1
    AOA-CDIND-1 [1-12]
    AOA-CDPAI-1
    FILLER
```

Example 3 (IMS) :

```
1 DBD     NAME=CUSMNGMT
2 SEGM    NAME=CUS,BYTES=35
3 FIELD   NAME=(CUSNB#,SEQ),BYTES=5,START=1
4 FIELD   NAME=ADDR,BYTES=30,START=6
5 SEGM    NAME=ORDER,PARENT=CUS,BYTES=20
6 FIELD   NAME=(ORDNB#,SEQ),BYTES=10,START=1
7 FIELD   NAME=AMOUNT,BYTES=10,START=11
```



Meta-modelization

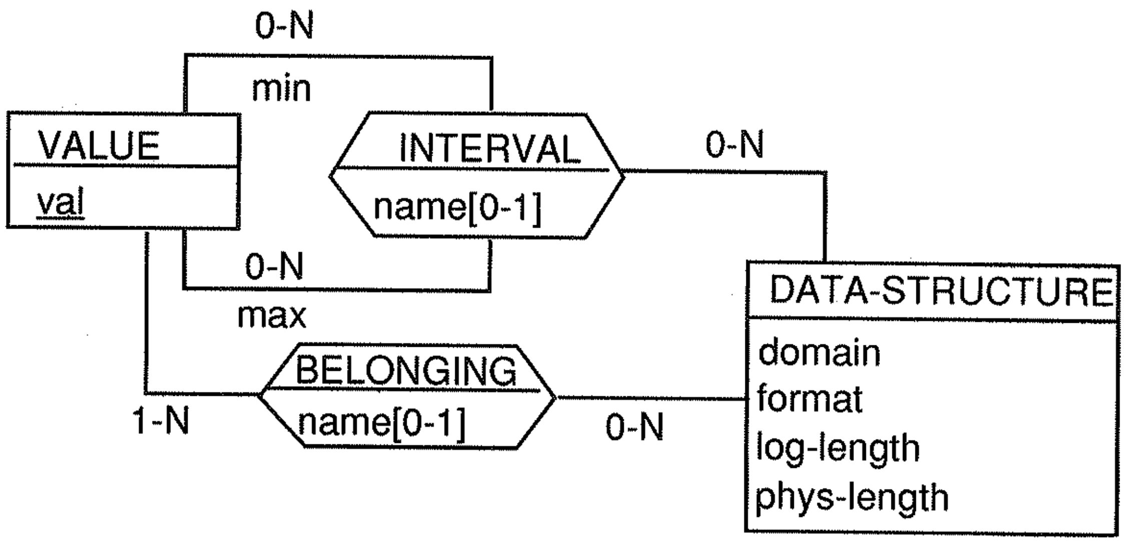


4. Typing concepts

- elementary data structure : explicit
  compound data structure : inferred
- **domain**
  Ex : "char" in SQL; "XXX" in COBOL or CODASYL
- **format**
  Ex : "S99V999" in COBOL
- **logical length**
  Ex : "char(10)" in SQL; "999" in COBOL
- **physical length** → **end position** for attributes
  Ex : "(packed decimal) numbers in COBOL are coded using one half-byte per digit".

- **intervals** : name, lowest value, greatest value
  Ex : COBOL :
  ```
  02 STATUS-VEHICLE PIC 9.
         88 NOT-OK VALUE IS 1 THRU 9 .
  ```
  PASCAL :
  ```
  NUMCLI : [1..1000];
  ```
- special **values** : value, name
  Ex : COBOL :
  ```
  02 STATUS-VEHICLE PIC 9.
         88 OK VALUE IS 0.
  ```
  PASCAL :
  ```
  color : (blue, red, yellow);
  ```

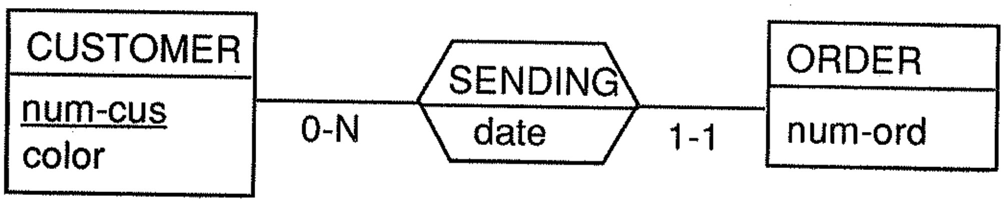
Meta-modelization



5. Constraints

- constraints → **strong-typed** : repetitivities
  cardinalities
  identifiers
  referential constraint
  roles exclusion
  . . .
- constraints → **weak-typed**

- **repetitivities**
  Example : COBOL/CODASYL : OCCURS
- **cardinalities**
  Example : CODASYL : mandatory/optional member
- **identifiers** : wide definition : A data structure can be identified by one or severals other data structure(s).
  - Example :



id(CUSTOMER)=num-cus
id(ORDER)=num-ord,CUSTOMER

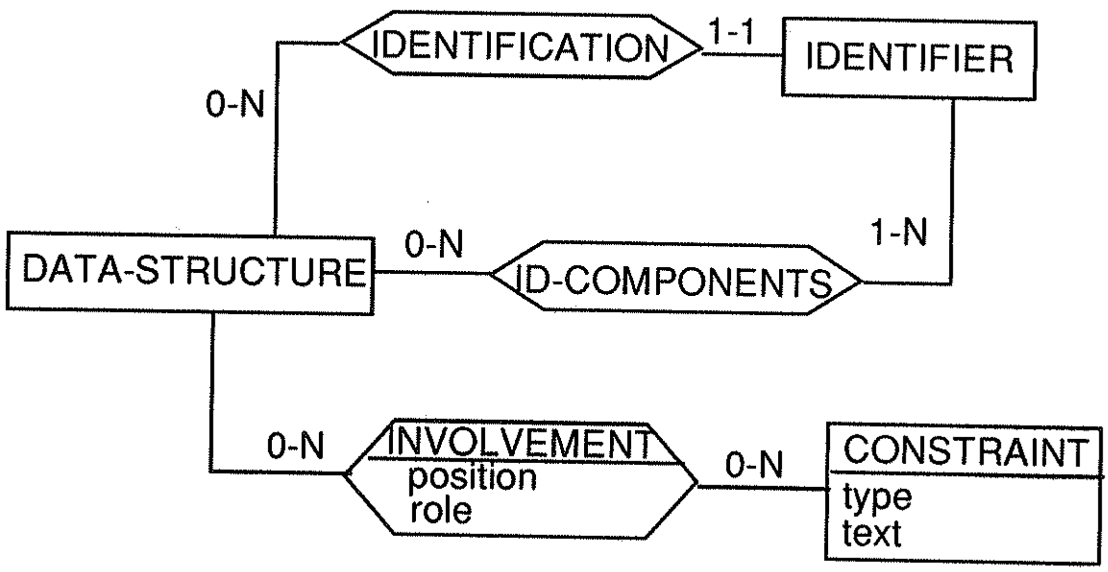
  - Physical examples :

COBOL	SQL	IMS	CODASYL
record key (without duplicates)	unique index	sequence field	calc key duplicates not allowed

Note : often a confusion with the access key concept.

- referential constraint
- weak-typed constraint : text, components
  Example :
  "for each VEHICLE/v :
  STATUS(v) = "free" <u>or</u>
  STATUS(v) = "being-repaired" <u>or</u>
  there exists a BORROWING/b : borrowed(b,v)"

<u>Meta-modelization</u>



6. Access structures

- **Access keys** : a fast mechanism to obtain instances of a data structure from (a) given value(s) of (one of) its attribute(s).

COBOL	SQL	IMS	CODASYL
record key	index	-	calc key

- **Sort keys** : one or several attributes which forms a sort criteria for a data structure.

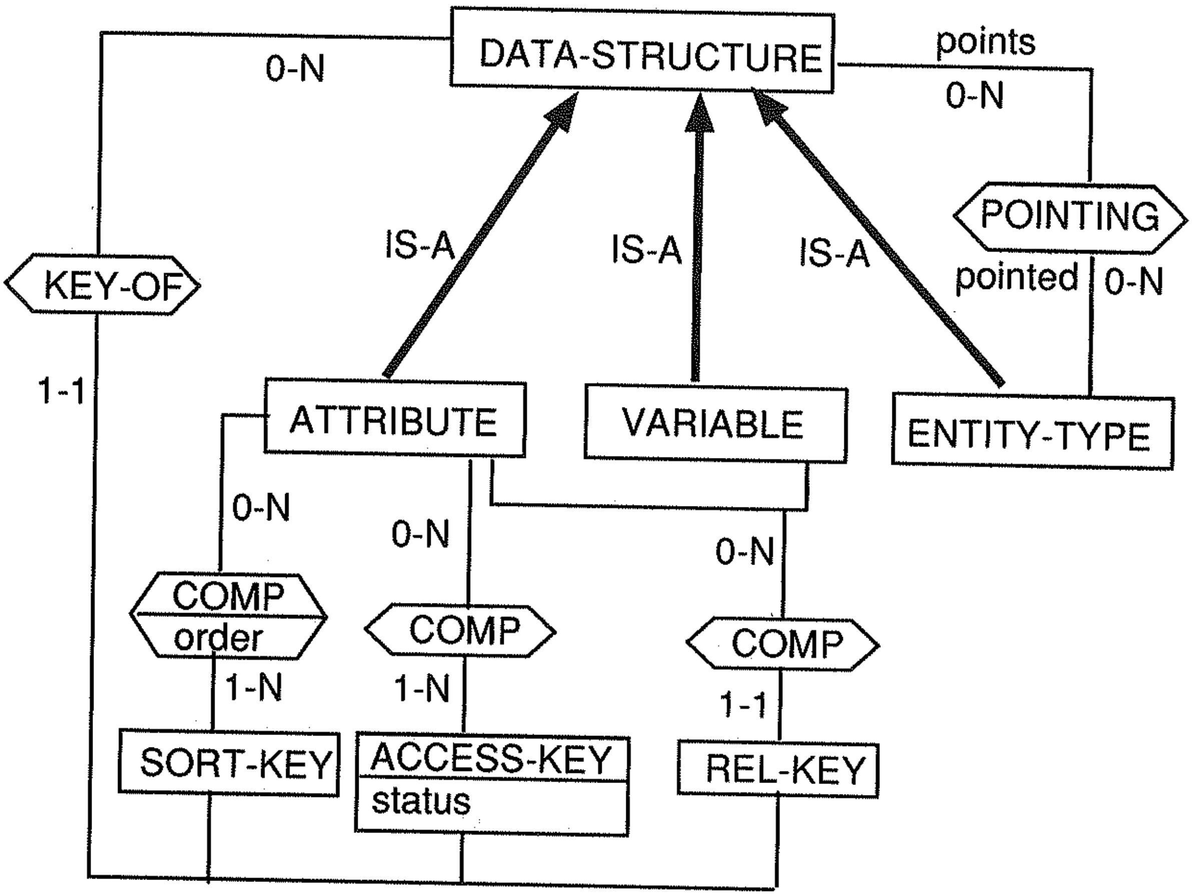
  Example : COBOL

```
SORT <file>
ON ASCENDING/DESCENDING KEY <donnee1>...
...
```

- **Relative keys** : a zone to record the current record number or get a relative access from it.

- **Pointers** : a data structure which points to an entity type.

Meta-modelization



7. Physical Structures

- **Source Text Files** : - file to be analyzed
  - inclusion relationship

- **Modules** : - any unit of processing
  - inclusion relationship

  Example : Cobol : program,
  (proc.div.) section/paragraph

- **Logical Files** : internal description of a physical file
  Example :

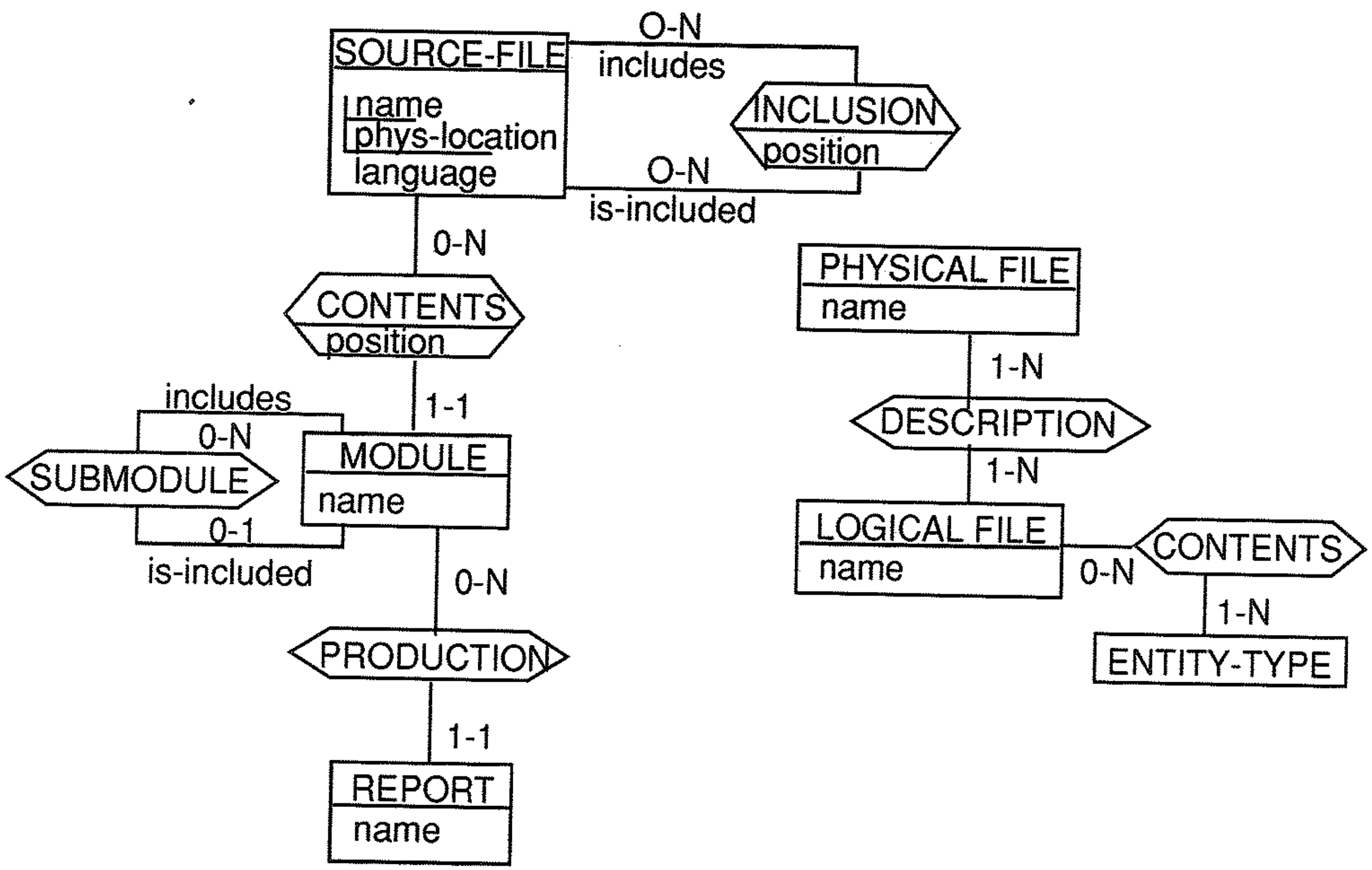
Cobol	Relational	IMS	Codasyl
logical file	schema	database	area

- **Physical Files** : physical area where data are stored.

- **Variables** : a variable used by a module.

- **Reports** : a report, form produced by a module.

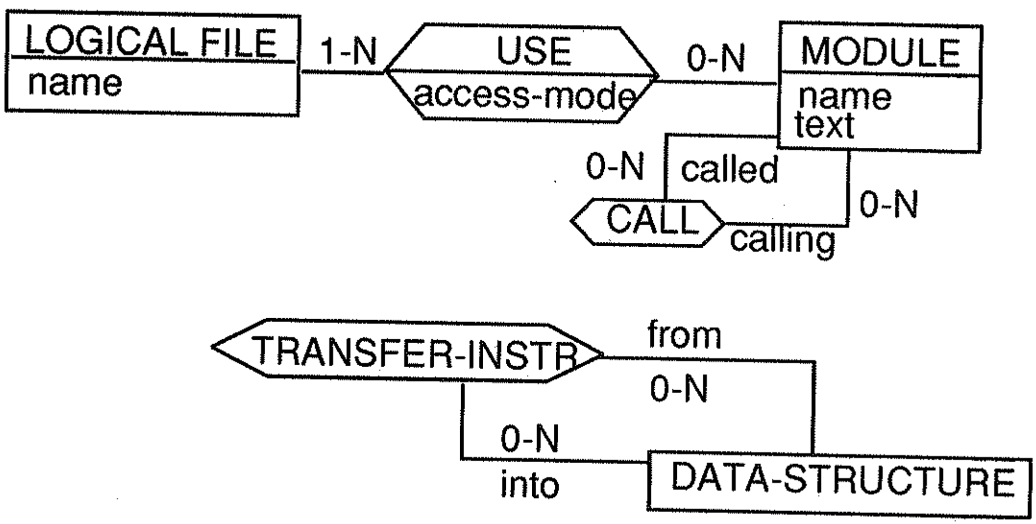
Meta-modelization



8. Statements

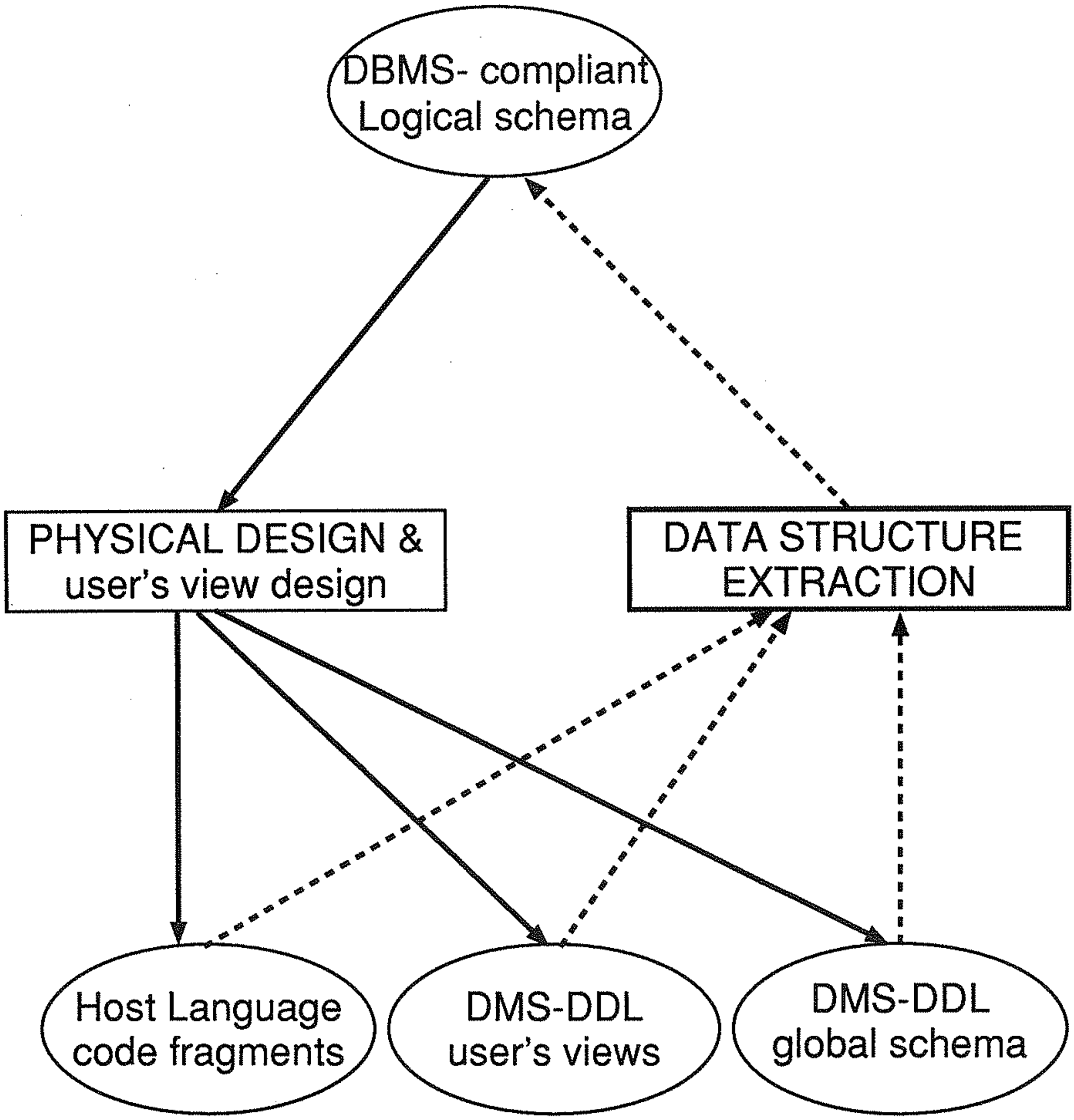
- Transfer instructions : any transfer between data structures.
  Example (Cobol) :
  - MOVE
  - parameter(s) binding : CALL USING
  - READ INTO
  - SORT/MERGE ... USING/GIVING ...
- Calls : a module calls another one.
  Example (Cobol) : - CALL
  - PERFORM
- Uses : a module uses a logical file with an access mode.
  Example (Cobol) : - OPEN INPUT
  - OPEN OUTPUT/EXTEND

Meta-modelization



Extraction

Schema


DBMS- compliant
Logical schema
PHYSICAL DESIGN &
user's view design
DATA STRUCTURE
EXTRACTION
Host Language
code fragments
DMS-DDL
user's views
DMS-DDL
global schema

Sources of information

Origin of information sources.

- Source code of the system.

  ex : source programs, libraries, data dictionaries, repositories.

  -- *most reliable* description of the system.

  -- *low level* description of the system.

- Higher level descriptions of the system.

  ex : development documentation : data-flow diagrams, state transition diagrams, E/R schema's, ...

  -- not always a complete, up-to-date or correct description of the system.

  -- high level abstraction of the system.

- Other information on the system.

  ex : entry forms, reports, users, developers.

Source code issues.

- Source code must be correct :
  - -- syntactical : the rules of the language are obeyed.
  - -- semantical : the system performs the functions it is designed for.

- Source code is complete :

  The source code is a compileable set.

- Different computer languages can be used in an application.

  ex : cobol + assembler + DDL of DBMS.

- If one language is used :
  - -- Differences due to different implementors.
  - -- Differences due to new versions and standards.

Preanalysis of application.

Objectives.

- Control the completeness of the application

- Get an idea on the current state of the application :
  - -- errors.
  - -- naming conventions.
  - -- programming conventions (structured).
  - -- dead code.

- Recover information for strategical purposes.
  - -- files accessed & mode.
  - -- components of application.

- Enhance application characteristics for reverse engineering.
  - -- removal of dead code.
  - -- restructuring of programs.
  - -- renaming conform to standards._

Methods & Tools

- Compilers and linkers.
  - -- check for syntactical correctness of application.
  - -- check completeness of application.

- Static analysis
  - -- *Code audit* : control code on syntax errors and conformity to standards.

    ex : suppression of 'GO TO' instruction, maximum nesting levels.
  - -- *Complexity analysis* : measurement of complexity of programs.

    degree of nesting, module size, number of records, number of data-items, number of comments.
  - -- *Flow analysis* :

    control flow analysis : detection of not used files.

    data flow analysis : relations between data items, unused variables,...

- Dynamic analysis
  - -- ensure that the system is doing what it is expected to do.
  - -- partially forward engineering practice.
  - -- usefull for old systems (not running anymore).

- Restructuring.
  - -- change program structure, without changing it's functionality.
- Removal of unused data and files.

Data extraction.

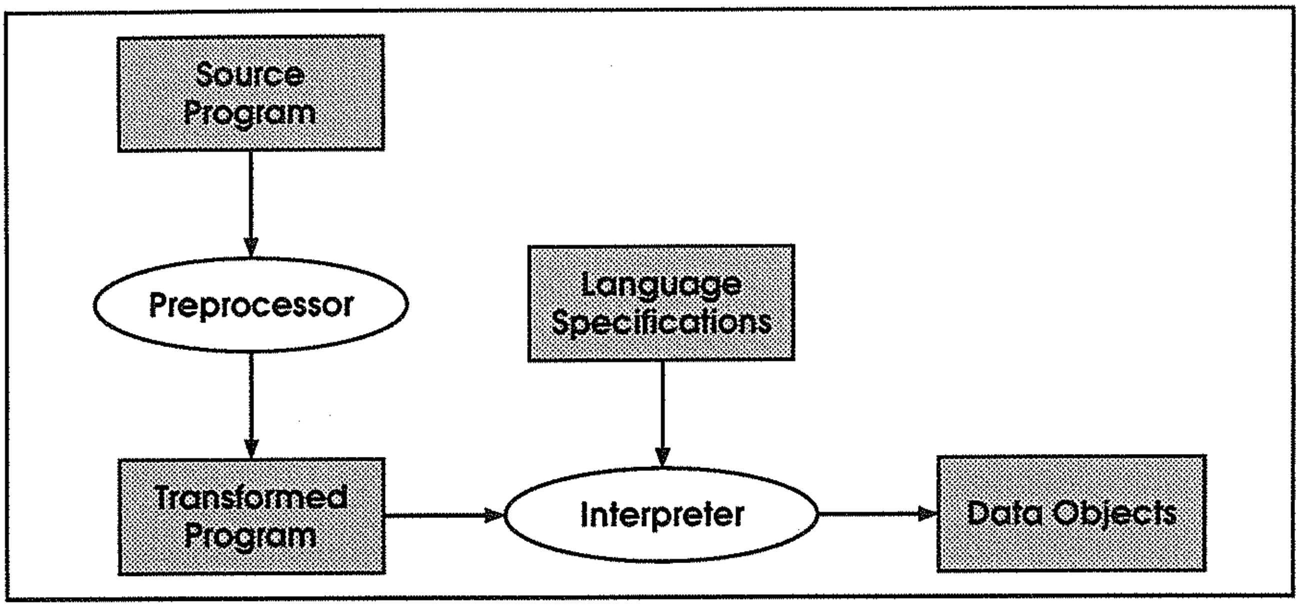
Objectives of data extraction.

- Recognition of "relevant concepts for reverse engineering" in the source code of the system.

- Translation of those physical concepts to those used at the level of the DMS compliant schema.

- Set a reference of the created objects to their position in the source code

A possible architecture for data extraction.



- The Language specifications contain :
  - -- Grammer definition of the source language.
  - -- Definition of the relevant data to retrieve.

  This will allow the analyser to be generic : It can be tuned to different source languages.

- The interpreter :
  - -- Searches the source program for relevant data.
  - -- Will create corresponding data objects.

Types of objects to extract.

Remark :

The objects which will be extracted from the source code are dependant on the type of DDL used in the source code.

Entity types.

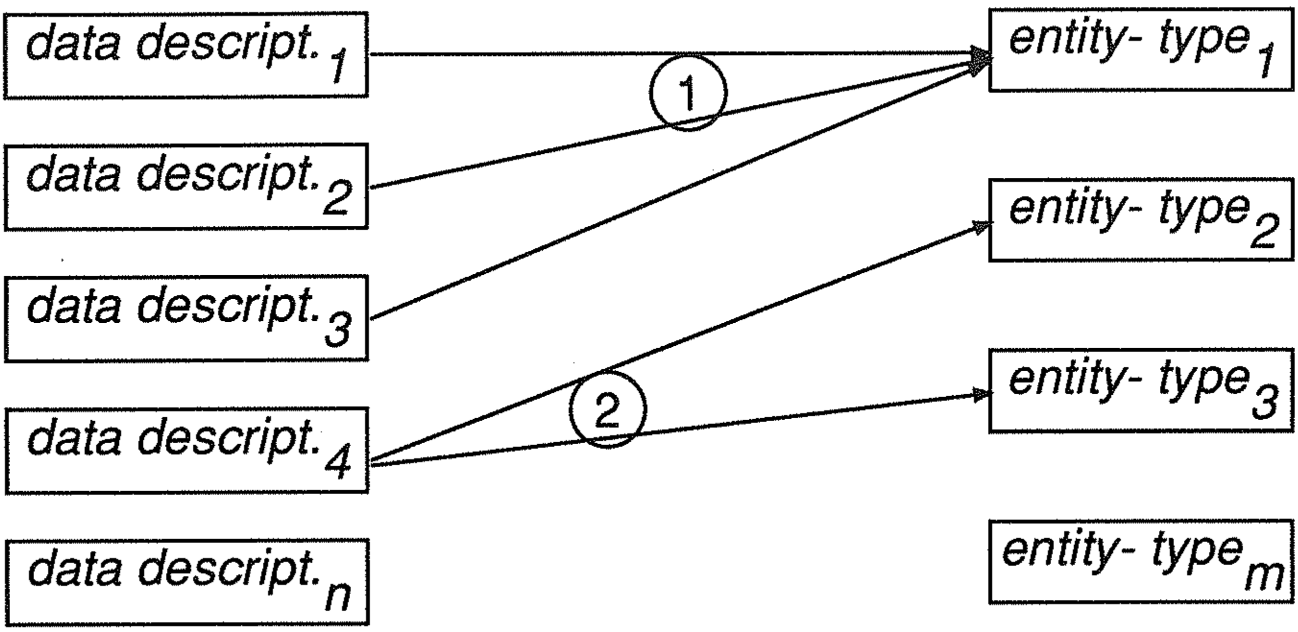
First approximation :

Data aggregate -> Entity type.

For example :

Cobol	record type	->	Entity type.
IMS	segment type	->	Entity type.
SQL	relational table	->	Entity type.

Problems in mapping :



(1) Overloading of datastructures : One data structure has multiple data-descriptions.

Ex : Redefines and renames in COBOL.

(2) One data description maps on several entity types.

Ex : Backup files, Overlapping applications.

Solution :

Field structure could reveal the detection of the entity types.

Finding attributes

First approximation :

Field description of Data aggregate	->	Attributes.

For example :

Cobol	record fields	->	attributes.
IMS	segment fields	->	attributes.
SQL	columns	->	attributes.

Problems :

1. Structure hiding :

Some structures or sub-structures do not provide a complete description.

ex : Usage of FILLERS in cobol record descriptions.

The *address* field of a cobol record type is not subdivided into it's components : *street* , *number* , *city*. *address*

street
number
city

Solution :

Check the data-flow of the application to retrieve a more detailed description.

ex :

Move *address1* to *address*

2. When has structure hiding been used

- Difficult problem to solve.
- Depends on application domain knowledge.
- Some heuristics can be used :
  - -- length & type of the field.
  - -- Usage of FILLERS (not at the end of a record description.)
  - -- Number of subfields of a record.

An example of entity and attribute extraction.

```
[BEGIN
 PATTERN    FD-RECORD ()
 ""
 DEFINITION
    $ {01,1} _ &record&
   ($ {01,1} _ &record&) ;
 DECLARATIONS
    &record&      -> &rec-name& [_]. (_&field&[_].);
    &field&       -> &level& _ &fld-name&
                           [_REDEFINES_&red-field&]
                           [_{PICTURE , PIC}[_IS]_&pict&]
                           [[_USAGE [_IS]]_&usage&]
                           [[_SIGN [_IS]] _{LEADING,TRAILING}
                                     [_SEPERATE [CHARACTER]]]
                           [_OCCURS_ &occurence& (_ &key-clause&)
                                    [_INDEXED [_BY] _&index& (_&index&)]]
                           [_{SYNCHRONIZED,SYNC} [_{LEFT,RIGHT}]]
                           [_{JUSTIFIED,JUST} [_RIGHT]]
                           [_BLANK [_WHEN]_{ZEROES,ZEROS,ZERO}]
                           [_VALUE[_IS]_&value&] ;

    &rec-name&    -> <variable> ;
    &level&       -> {0 2-9 , 1-4 0-9 , 2-9 } ;
    &fld-name&    -> {FILLER , <variable>} ;
    &red-field&   -> <variable> ;
    &pict&        -> <picture> ;
    &usage&       -> {COMPUTATIONAL,COMP,DISPLAY,INDEX} ;
    &occurence&   -> {&min-occ1& _TO_ &max-occ& [_TIMES]_DEPENDING
[_ON]_&occ-var&,
                      &min-occ2& [_TIMES]};
    &min-occ1&    -> <n-lit> ;
    &min-occ2&    -> <n-lit> ;
    &max-occ&     -> <n-lit> ;
    &occ-var&     -> <variable>;
    &key-clause&  -> &asc-desc& [_KEY] [_IS]_&occ-key& (_&occ-key&);
    &asc-desc&    -> {ASCENDING,DESCENDING};
    &occ-key&     -> <variable>;
    &index&       -> <variable>;
    &value&       -> <lit> ;
 ACTION
   $(forall &record&
     (create_record));


  END]
```

Relationship types

This applies only for DMS which allow for their representation :

ex :

IMS	hierarchical link	->	relationship type
IDMS	set type	->	relationship type

The cardinality :

IMS	one-to-many.
IDMS	one-to-many or many-to-many

For those DMS which do not provide a concept for representing relationship types ex : relational systems , COBOL systems a transformation has been applied :

reference field + referential constraint.

Access Keys

- technical construct (index , hash files).

- is important to describe the database.

- they can give indications on the existance of :

  -- identifiers : if no duplicates are allowed.

  -- relations : reference fields are declared as an access key.

Finding identifiers

- Strongly dependent on the type of DMS.

  ex : RECORD KEY.

  ALTERNATE RECORD KEY WITHOUT
  DUPLICATES

- Some identifiers may be found by examening the program text.

Other integrity constraints.

- Referential integrity constraint
  - -- Some DMS allow for their specification

    ex : SYBASE usage of triggers.
  - -- Otherwise hidden

    name structures, modification structures in procedural parts, event driven entry forms.

- Functional dependencies, structural redundancies, exclusive attributes/roles value dependancies
  - -- Some RDBMS allow for their declaration (performance)
  - -- Other hidden structures

Integration process

(The multiple view problem)

Contents :

- Situation
- Proposed model within the PHENIX - research
- The several steps of an integration process
- The retrieval of semantic corresponding parts
- Theoretical explanation of the integration
- Some examples

Situating the multiple view problem / integration problem :

- ideal conceptual schema :
  - -- no redundancy,
  - -- no multiple views of the same database

- problem:
  - -- when large sets of data structures are involved; avoidance of redundancy is not always realistic:
    - > optimization reasons
    - > division of the work in subparts
    - > ....

==> controlled redundancy is required

- Integration within Forw. Eng.
  <--> Integration within Rev Eng

  -- Forw. Eng. :
    > Integration of multiple views to obtain one global solution.
    > Integration is considered at the conceptual level
    > ....

  -- Rev. Eng. :
    > Integration is considered at several levels
      # physical level
      # logical level
      # conceptual level

    > Integration is considered not only between multiple views but also within one and the same view

- Consequence:

  -- data structures have to be considered at several levels of description

  -- a same fact can be represented with several representations within an E/R model

  ==> a specific interpretation model of the E/R model will be used : a semantic network

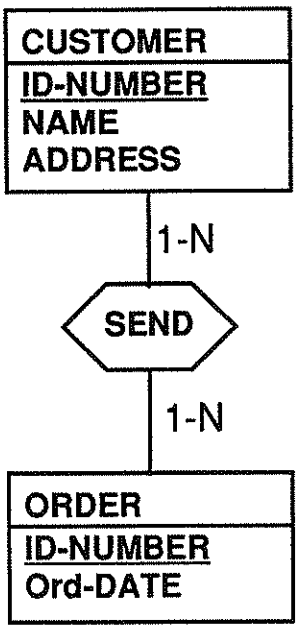
Proposed model :
SEMANTIC NETWORK.

- **Two structures are considered:**

  classes

  paths

Example :




```
01 ORDER1
   02 ORD-NUM          PIC X(8)
   02 ORD-DATES        OCCURS 2 TIMES
                       PIC X(6)
   02 CUS-NUM          PIC X(7)
   02 CUS-STREET       PIC X(15)
   02 CUS-NB           PIC X(3)
   02 CUS-ZIP          PIC X(4)
   02 CUS-CITY         PIC X(20)
```

- classes:
  - -- the nodes of the semantic network
    - > entity types
    - > relationship types
    - > attribute types
    - > fields of a record description

examples:

CUSTOMER, ORDER, SEND,
ID-NUMBER, Ord-DATE,
ORDER1, ORD-NUM, CUS-NUM, ...

- Arcs:
  - -- special case of paths (see later)
    - > roles of a relationship type
    - > relationships between an attribute (or a record description field) and its father

examples:

CUSTOMER<-->SEND, SEND<-->ORDER,

CUSTOMER<-->ID-NUMBER,

ORDER1<-->CUS-NUM, ..

-- An arc has also cardinalities (in both directions)

> attribute-arc:

father->attribute : repeating factors of the attribute

attribute->father : 1-1 (identifying attribute) or 1-N

> role-arc:

reltype->enttype : 1-1

enttype->reltype : cardinalities of the role

examples:

CUSTOMER->SEND	: 1-N
SEND->CUSTOMER	: 1-1
ORDER1->ORD-DATES	: 1-2
ORD-DATES->ORDER1	: 1-N

- PATHS :
  - -- a list of arcs with intermediary classes

example:

ORDER<-->SEND<-->CUSTOMER<-->ADDRESS

-- A path has also cardinalities which are calculated from the cardinalities of the intermediary arcs
(minimum of minima and
maximum of maxima)

The several steps of an integration proc.

- **input :**

  two structures which have to be integrated (or one structure has to enrich the other, ...)

- **output :**

  one structure containing all the semantical constructs available within both original structures

- **intermediary tasks :**

  -- finding corresponding parts

  -- integration of the main original structures

  -- integration of the child structures of these original structures

  -- integration of the arcs / paths of the structures

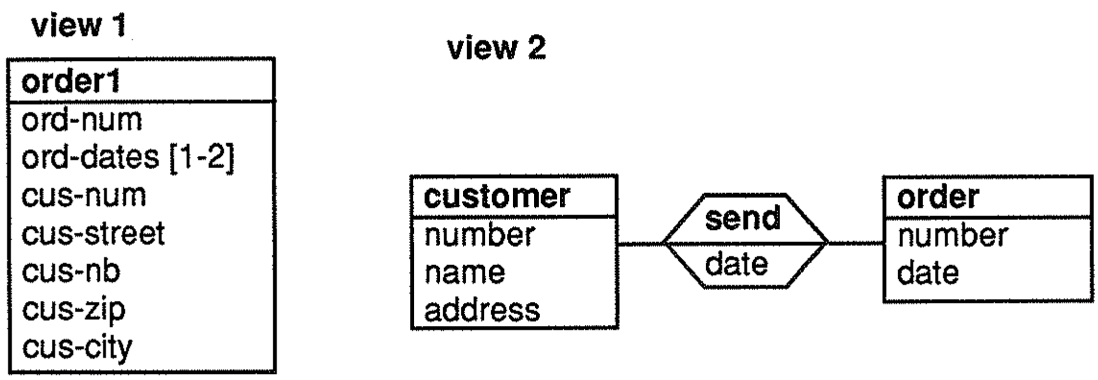
The retrieval of corresponding parts

Before a structure can be integrated with another structure it is required to find the corresponding part of a structure.

There are however several types of correspondences between structures and there are also several techniques wich can be used to detect corresponding parts.

types of correspondences:

1. two structures are identical:



order1 and order are identical structures

2. **A structure is a generalization of another structure**
   or
   a structure is a specialisation of another structure
   example :
   A. an employee is a special case of a person
   B. a luitenant is a special case of military personnel
   C. ...

3. **A structure has no real correspondence but there is another structure such that both structures are special cases of another unknown third structure**
   example :
   MAN and WOMAN are special cases of a third structure:
   PERSON

4. A single structure corresponds to an aggregation of other structures

example :

attribute address <-->

attributes street, nr, city, zipcode

5. A structure which represents a list of facts (grouping structure) can correspond to several structures representing the facts individually

example :

dates[1-3] <-->

birth-date, date of marriage, date of dead

How to detect equivalent parts

Several detection techniques are possible :

1. some are more automized then others
2. some are usefull every time, others are specific to a certain context.
3. most (if not all) detection techniques are uncertain.

   ==> several detection technique have to be considered together, confirmation has to be given by the user, ...

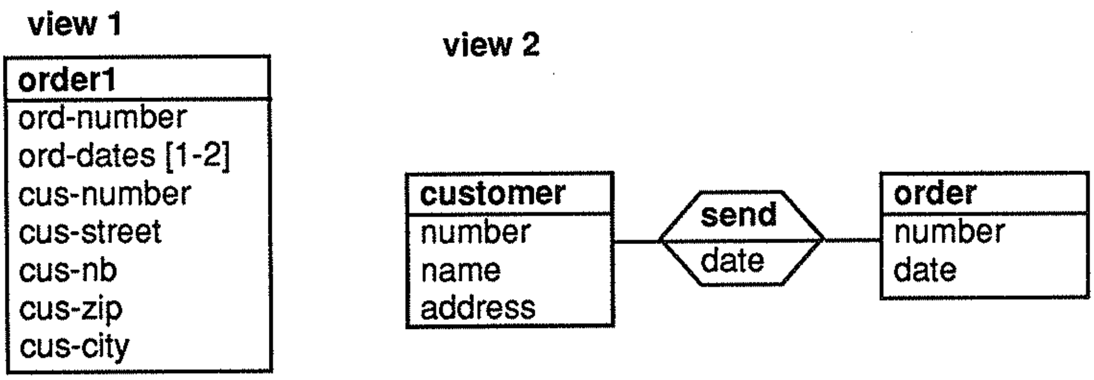
Usefull techniques & remarks :

1. Name comparison

useability :

relationship type, entity type

attribute: less significant technique unless the search-space of possible corresponding parts is reduced.



entity types : ORDER1 ~~ ORDER

==> attribute type order1.ord-number ~~ order.number

how to compare :

A. equality of names :

string-comp. to find classes with equal names

string-comp. to inspect similarity of substrings

...

B. semantical equality of names :

much more difficult : the system has to know which names are synonyms.

In the case of a multi-language enterprise the translation of a name has to be inspected too.

==> name comparison is much more than simple string-comparison !!

Remark:

homonyms may cause problems during the comparison of names

2. length and format comparison

If two classes are equal, it is required that the length of their fields, the format of there fields, ... is equal.

REMARK :

A. Within an application there may be however multiple (non-conflicting) formats of the same class.

e.g.:

the first programmer uses PIC IS 99

the second programmer uses PIC IS XX

==> not necessarily a conflict !

B. If we consider two entity types PERSON.

PERSON1
ID-NUMBER NAME TELEPHONE ADDRESS MARR.-STAT.

PERSON2
ID-NUMBER NAME TELEPHONE ADDRESS

useability :

Especially in the case of attribute comparison.

Especially in the case of physical integration.

==> compare also position in a record !!

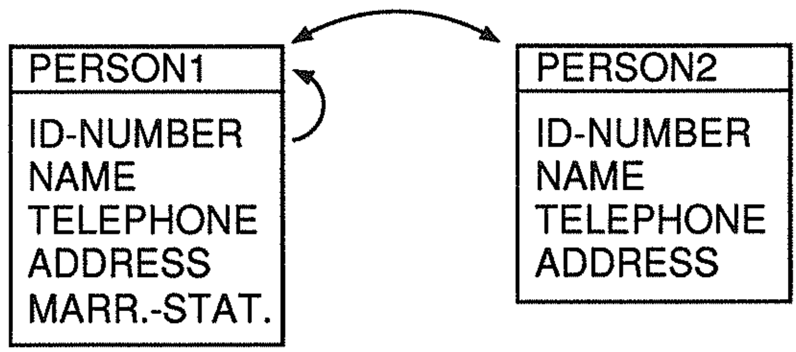
3. Context comparison

useability :

Conceptual / logical level :

Compare classes by looking at the correspondence of there nearest structure.

Example :



nearest structure of PERSON1.ID-NUMBER is PERSON1
PERSON1 is correspondent to PERSON2
==> context to find correspondent of
PRODUCT1.ID-NUMBER == atts of PERSON2

Physical level :

An important hint is the data memory space described by these structures

Examples :

A. different record descriptions of the same logical file

B. different record descriptions of the same physical file (several logical files with the same physical file).

C. transfer instructions between two variables ==> candidates to be integrated

D. REDEFINITION and RENAMING clauses within COBOL.

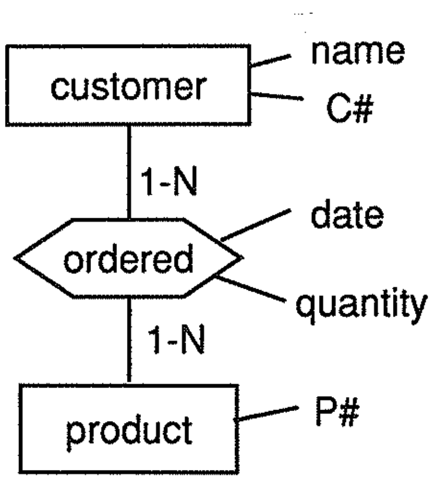
Example:

```
01 EPSVAL
   02 DATE-VI PIC 9(8).
   02 VI-DDMMYY REDEFINES DATE-VI.
      03 DD PIC IS 99.
      03 MM PIC IS 99.
      03 YY PIC IS 9999.
   02 CODVAL PIC IS 9(9).
   02 VI PIC IS 9(6).
   02 FILLER PIC IS X.
```

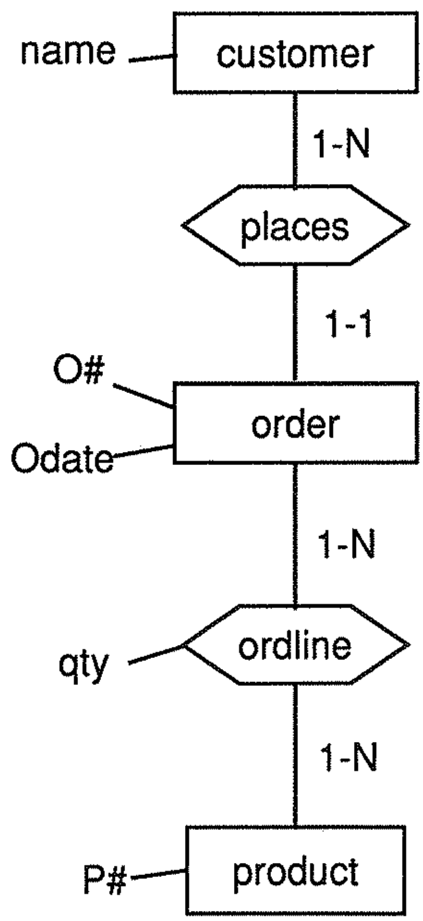
Path / Path equivalence :

precondition: classes linked by the paths are already considered

consequence : list of all arcs/paths linking the corresponding classes of the original ones will be searched for a corresponding arc/path.



ordered.date = Order.Odate
ordered = ordline
==>
? ordered - ordered.date
= ordline-order-order.odate ?



The integration of two views / structures

basic principles :

1. Every class and arc of both views must be present in the resulting view, whatever its form.
   A. if a structure is present in only one view
   ==> the original form will be kept and be available within the integrated version
   B. if a structure is available in both views
   ==> only one form must be retained

2. The approach is object oriented.
   ==> the child classes of a father class will be considered independent of the father class.

3. To be able to integrate an arc / a path it is required that there is a resulting integration of all the classes of the path
   ==> the arc integration will be considered in the framework of the related classes

   relationship type + roles
   attribute type + link to the father.

The integration of equivalent structures.
(See methodological manual Chapter 10, paragraph 7)

1. **A class has no correspondent :**
   A. Entity types will simply be added to the result
   B. New attributes will simply be added to the resulting integration of the father ==> the arc is also integrated

      example:

         ord-qty

         will simply be added to the integrated version

         of the father order
   C. New relationship types will be created with new roles equivalent to the original ones.

2. **An arc has no correspondent :**

   ==> a semantic link will be created between the integrated versions of the classes linked within the original version

   A. The classes to be related or ETs ==> create a RT

B. One class is an attribute: An attribute has only one father ==> the creation of an arc to a second father requires a transformation of the attribute into an entity type.

C. One class is a relationship type ==> an additional role will created if possible otherwise the RT is transformed into an ET.

3. Class / Class integration :

==> one class, in one of its original forms will be taken

proposal : consider the integration process as an enrichment process ==> the first class will be used to enrich the second class.

Take at first instance the class type of the enriched class within the integratedschema. Only if one of the arcs requires it the type of the class will change.

4. Class / Classes integration :

We consider two cases

A. attribute <--> set of attributes

==> use simple attribute to transform set into aggregation

B. grouping attribute <--> set of attributes

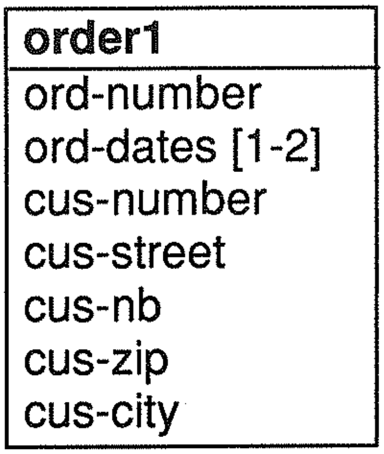
==> choose one of both used as preferred by the user

5. Arc / Multi-arc :

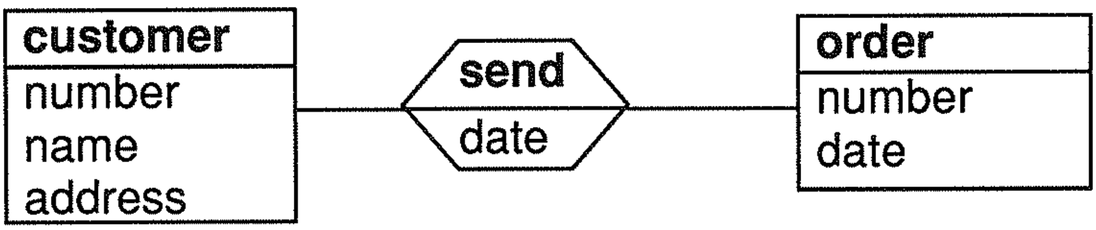
the path contains more detailed structures

==> choose the path

view 1


order1
ord-number
ord-dates [1-2]
cus-number
cus-street
cus-nb
cus-zip
cus-city

view 2


customer
number
name
address
send
date
order
number
date

The example:

Consider that view2 is used to enrich view1

1. ET ORDER2 + ET ORDER1 ==> ET ORDER
2. ORDER1.qty has no correspondent
   ==> create attribute qty in ORDER
3. subpart of ORDER2: number +
   attribute number of ORDER2
   ==> attribute number of ORDER
4. order.date-fur + send.date = order1.dates
   retard integration because send is not yet
   considered
5. ET CUSTOMER has no correspondent
   ==> create ET CUSTOMER in view1
6. the subpart customer.name has no corresponding
   ==> create subpart name of Customer
7. corresponding parts of
   customer.number, customer.address are respectively
   order1.cus-num and order1.cus-address

8. arc / multi-arc correspondence:

   Order1 -Order1.cus-num <-->

   order-send-customer-customer.number

   Order1-Order1.cus-address <-->

   order-send-customer-customer.address

   The path will be retained

   ==>choose intermediary class send too

   Create send because there is no correspondent

Transformation processes

Introduction

Definition

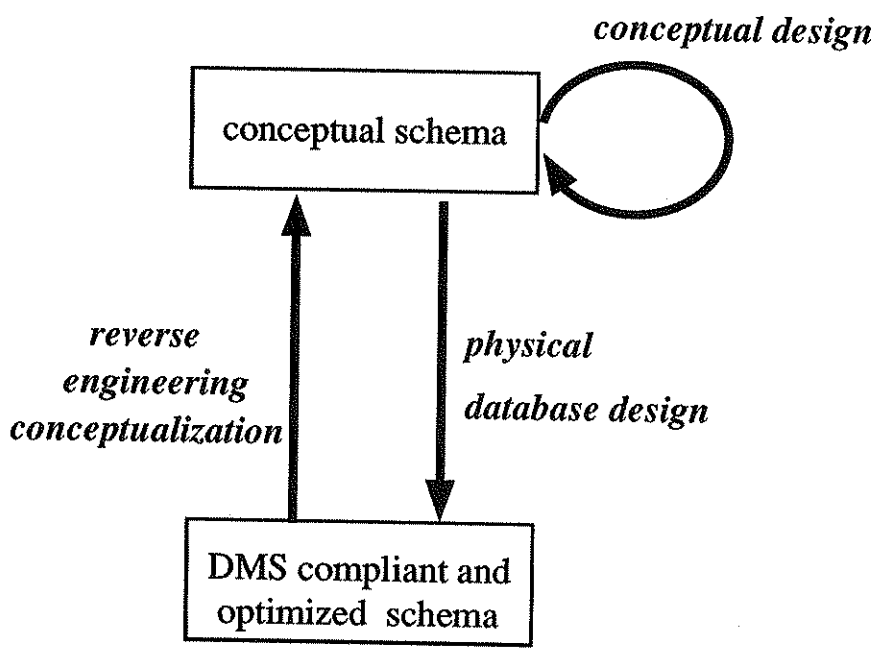
"transformation consists in applying structural modifications to a schema keeping as much as possible its semantics unchanged"

Objectives

•**Conceptual design.** Semantics-preserving transformations allow the increase/decrease the degree of expressivity of a schema and to improve its overall qualities (promoting/demoting transformations)

•**Physical data base design.** The goal is to adapt conceptual structures to poorer structures of a DBMS compliant-schema and to optimized it.

•**Reverse engineering conceptualization.** The transformations help to "decode" physical structures and to rexpress them in a more conceptual and understandable

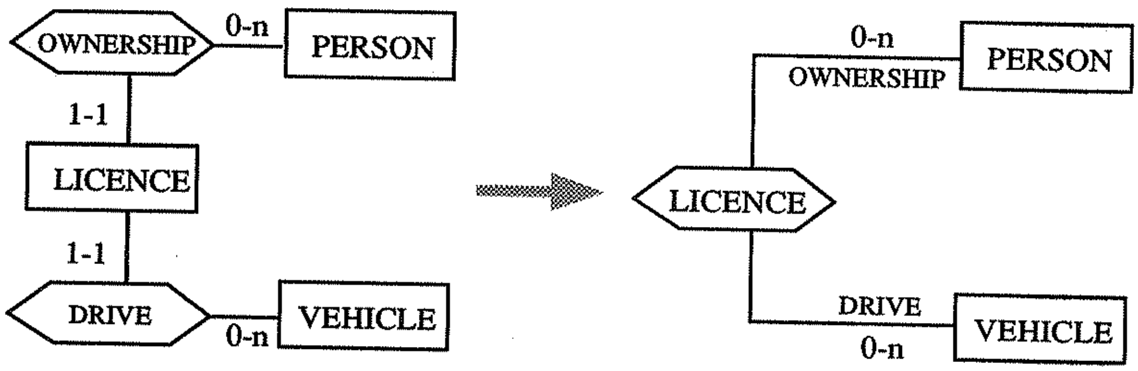


Entity-type into a relationship-type

Principles

An entity type which is related with other entity types (at least two) by 1-N relationship types only is transformed into a relationship type.

Example

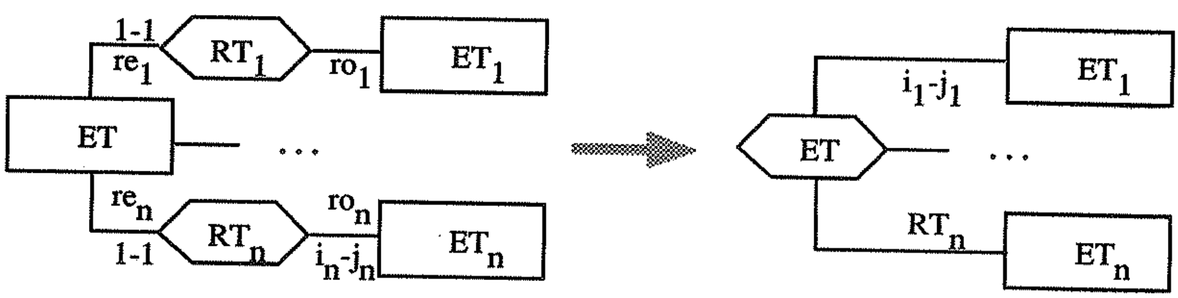


Goals ***conceptualization (demotion)***

Triggering situations

- An entity type has few attributes (date is often a typical attribute of a link)
- An entity type has only two or three 1-N relationships.
- An entity type has a name which is the concatenation of some entity types which it is linked to.
- The name of an entity type is a verb
- An entity type has not a local identifier, made up of own attributes only. Some components of its identifier are roles.
- The semantics of real-world objects described by an entity type is closer to links than to independent objects.

Definition



Preconditions :

- ET , the main data-structure must have at least two link-data structures
- All the link data structures RT_i are 1-N (more precisely the cardinalities of the participation re_i must be 1-1), binary, non-cyclic, without any char-ds

  Note : ro_i can be multiple
- rei cannot be multiple
- ET must not be nor a specific nor a generic main data structure

Actions :

1. Replace the entity type ET by a relationship type with the same name, the same attributes and the same identifiers/keys
2. Each relationship type RT_i of ET is replaced by a role with :

   a) the name of RT_i

   b) the same cardinalities as ro_i

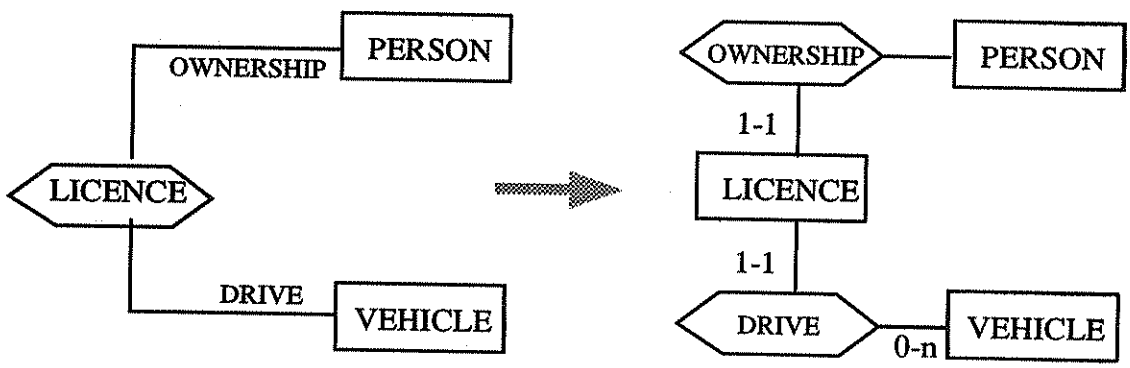
   c) the same identifier participations as ro_i

Relationship-type into entity type

Principles

A relationship type is transformed into an entity type, its roles being transformed into relationship types.

Example



Goals

conceptualization (promotion)

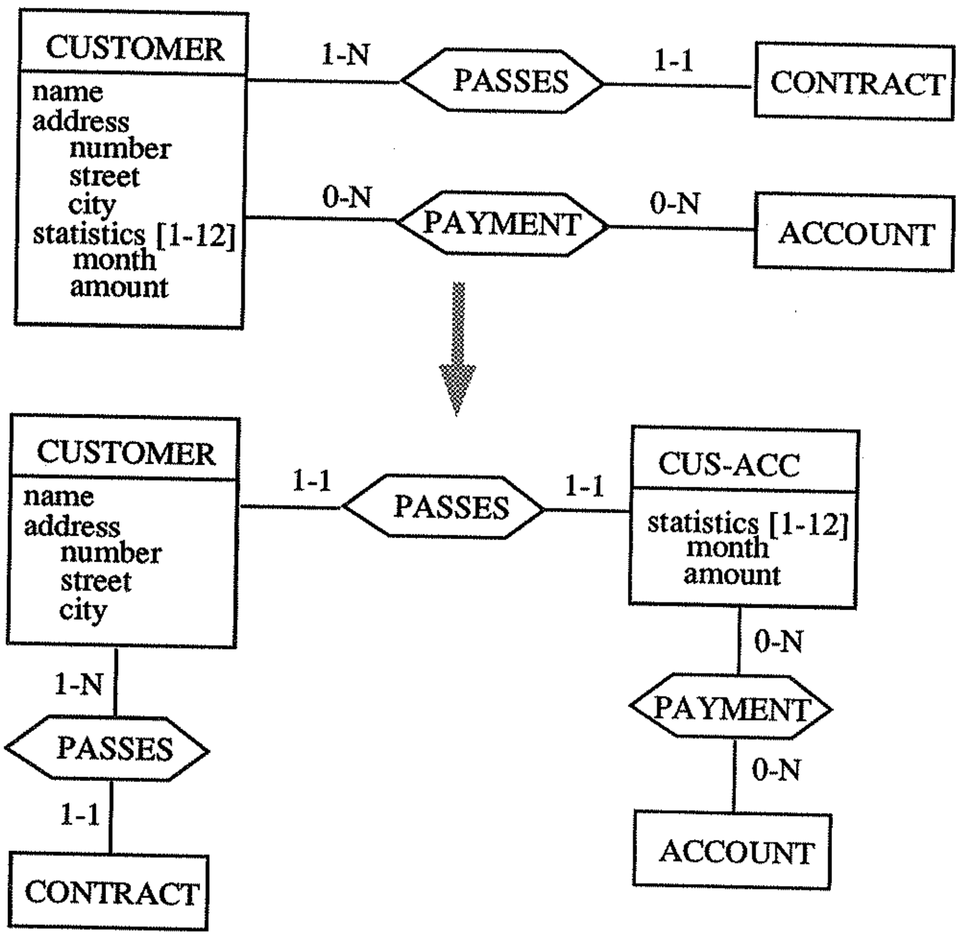
- In **conceptualization**, this transformation allows the promotion of a relationship type into a more important and independent concept, which new relationship types could be defined on.

Splitting of an entity-type

Principles

The attributes and roles of an entity type are distributed into two new entity types replacing the first one. These two entity types are linked by a 1-1 relationship type.

Example



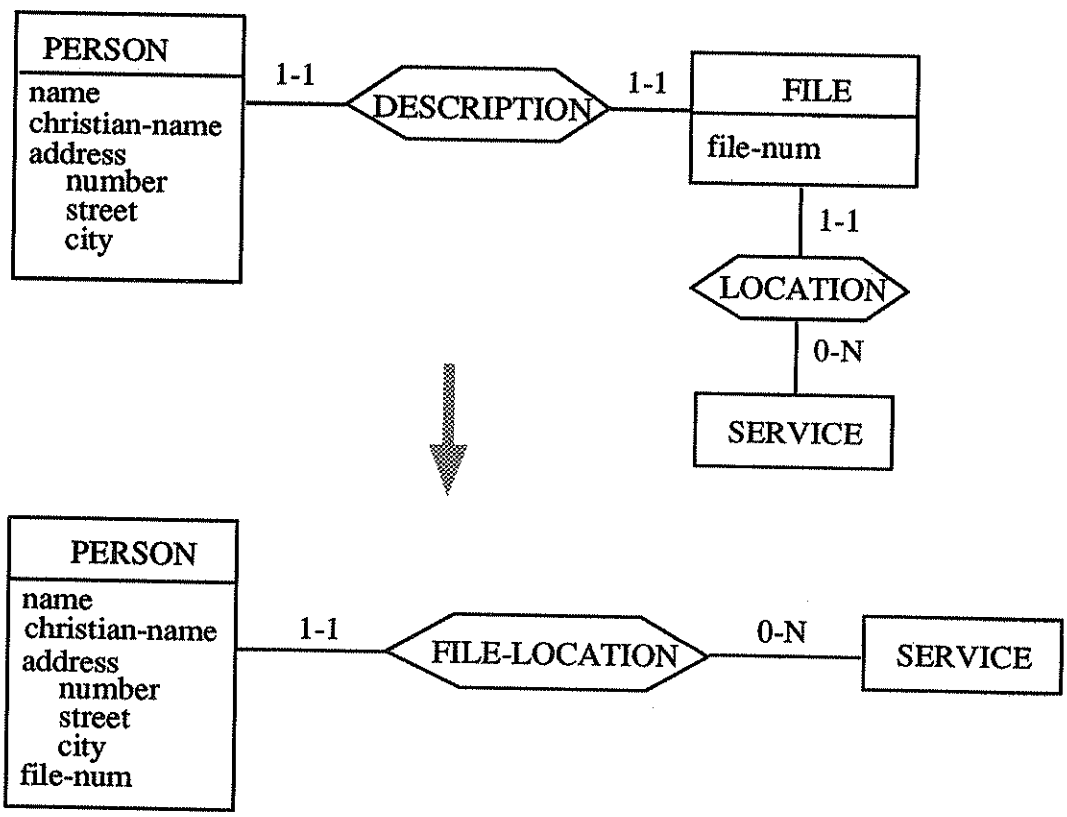
Goals RE conceptualization

Merging of several entity-types

Principles

Two entity types linked by a 1-1 relationship type are merged into a single one, which inherits all the attributes and roles of these two entity types.

Example



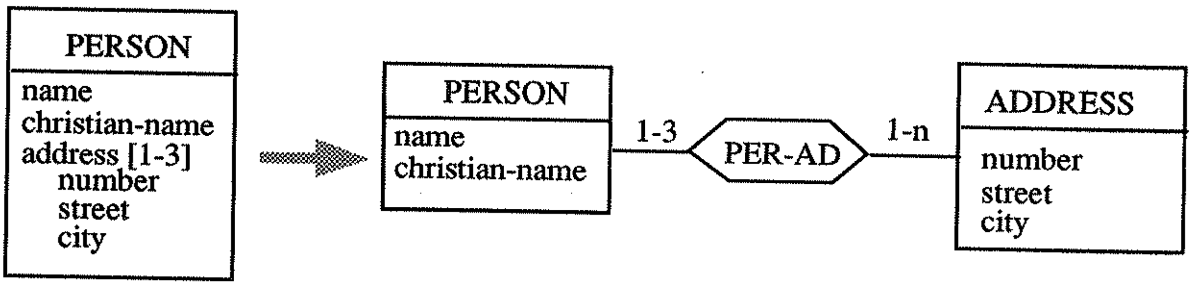
Goals **RE conceptualization**

Attribute into entity-type

Principles

An attribute of an entity type is transformed into an entity type, which is related by a new relationship type to its original father.

Example



Goals conceptualization (promotion)

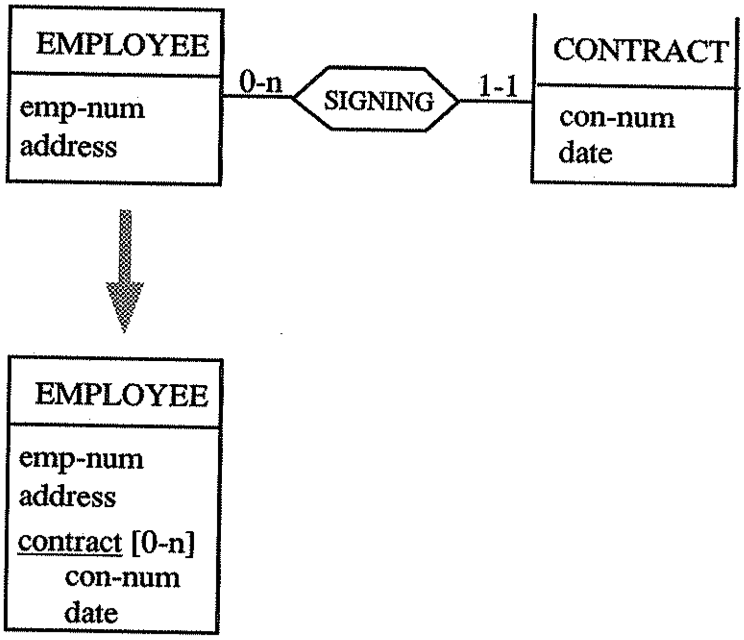
- in **reverse engineering conceptualization**, it can be used to split too much aggregated record types (frequent in file management systems, such as Cobol).
- in **conceptualization**, the attribute is **promoted** into a more important and independent object, which new relationship types could be defined on.

Entity-type into attribute

Principles

An entity type which is related with one and only one entity type is transformed into an attribute of this entity type.

Example



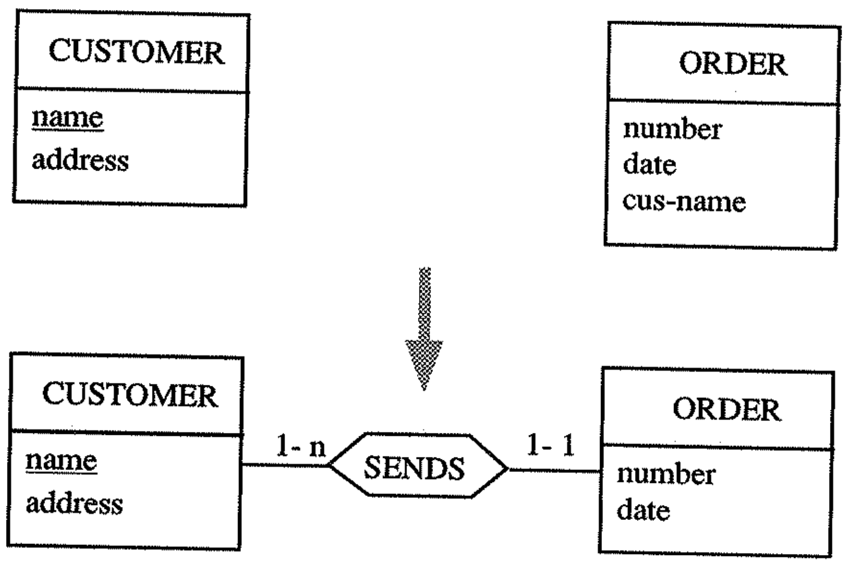
Goals conceptualization (demotion)

Referential attributes into relationship-type

Principles

One or several attributes of an entity type, which references an entity type (there is an inclusion constraint from these attributes to the identifier) are replaced by a relationship type linking these two entity types.

Example



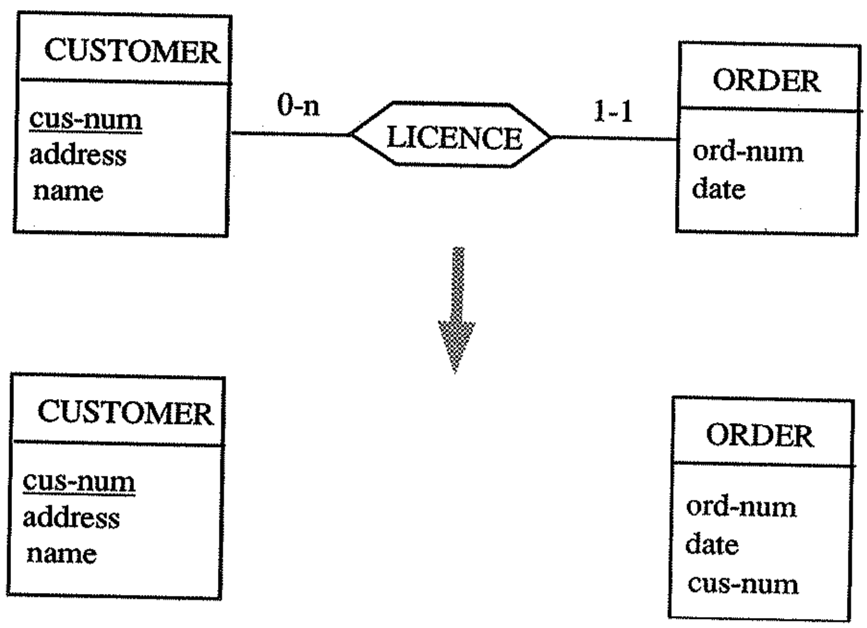
Goals **Re conceptualization**

Relationship-type into referencial attributes

Principles

A 1-N relationship type between two entity types is replaced by a reference, made up of attributes, in the entity type on the N-side to the entity type on the 1-side.

Example



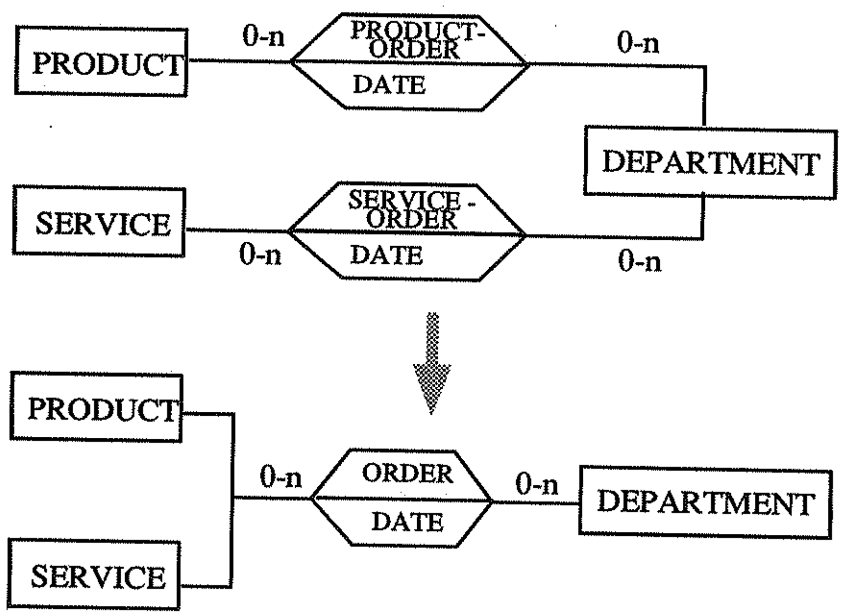
Goals Physical database design

Merging several roles of similar relationship-types into a multidomain role

Principles

Several relationship types which are very similar, except the entity type(s) participating to a role, are merged (generalized) into a single one.

Example



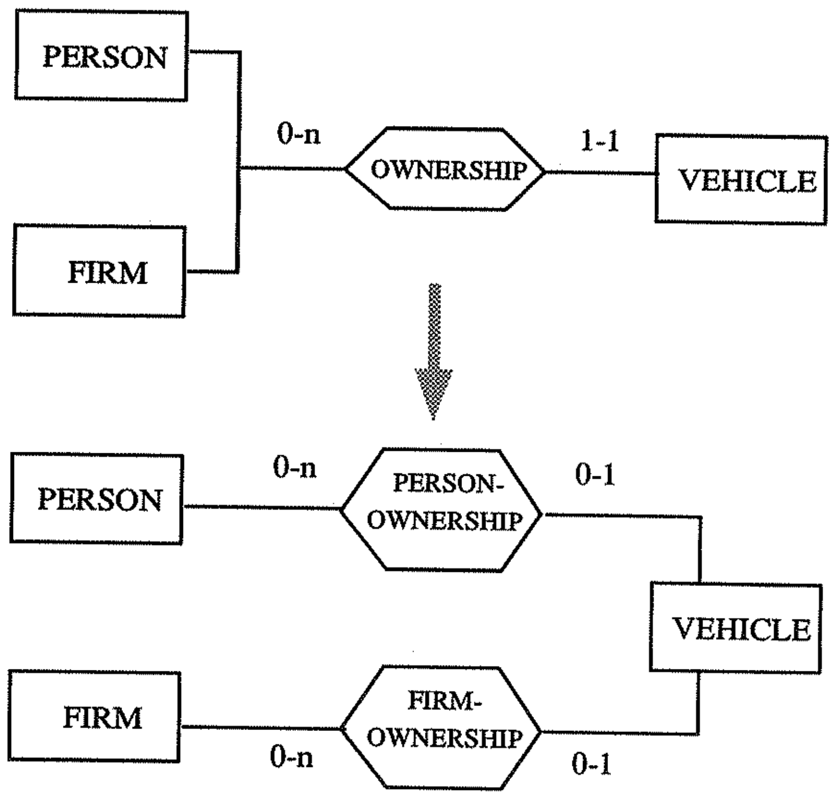
Goals Conceptualization

Split multi-domain role into several roles and relationship-types

Principles

A relationship type containing a multidomain role is split into several similar relationship types, one for each entity type participating in this role.

Example



Constraint : ∀ v ∈ VEHICLE :
there must be 1 and only 1 PERSON-OWNERSHIP,(or) FIRM-OWNERSHIP

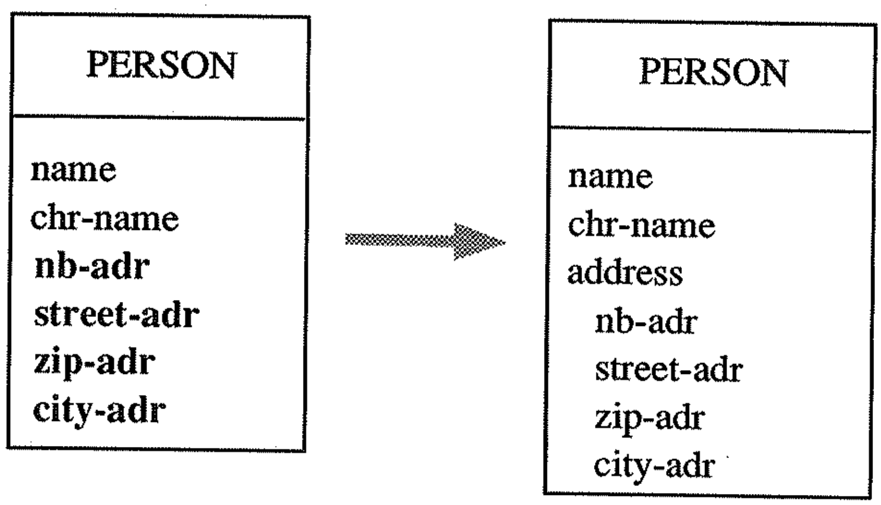
Goals conceptualization (demotion)

Aggregate a list of attributes into a compound father attribute

Principles

A list of attributes are aggregated into a new compound attribute. They are therefore redefined one level lower.

Example



Goals

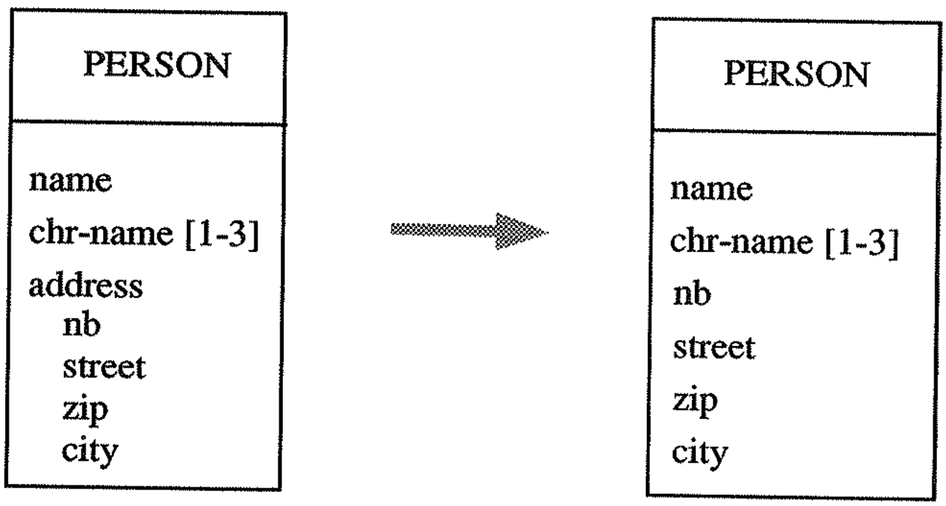
RE conceptualization

Replace a compound attribute by its components

Principles

A compound attribute is replaced by its component attributes, which are therefore redefined one level higher. The aggregation link between these attributes is lost.

Example



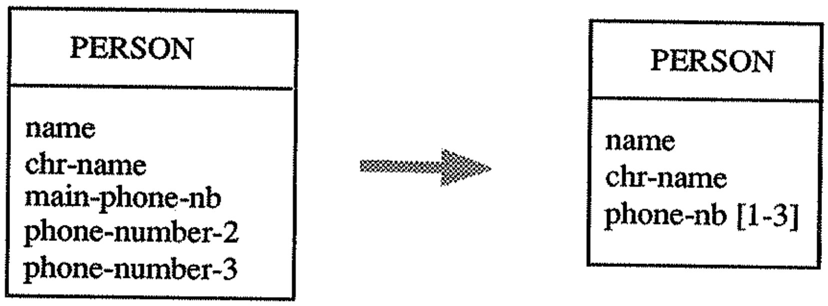
Goals **physical database design**

List of similar attributes into a multivaluated attribute

Principles

A list of similar attributes are replaced by a single multi-valued attribute, which groups all the values of these attributes.

Example



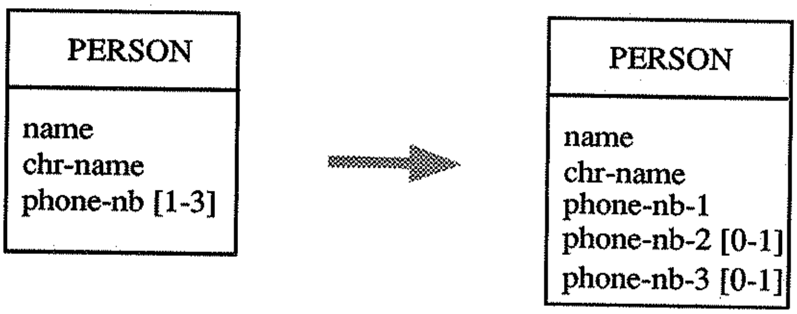
Goals   RE conceptualization

Multi-valued attribute into a list of attributes

Principles

A multivalued attribute is replaced by a list of similar simple attributes.

Example



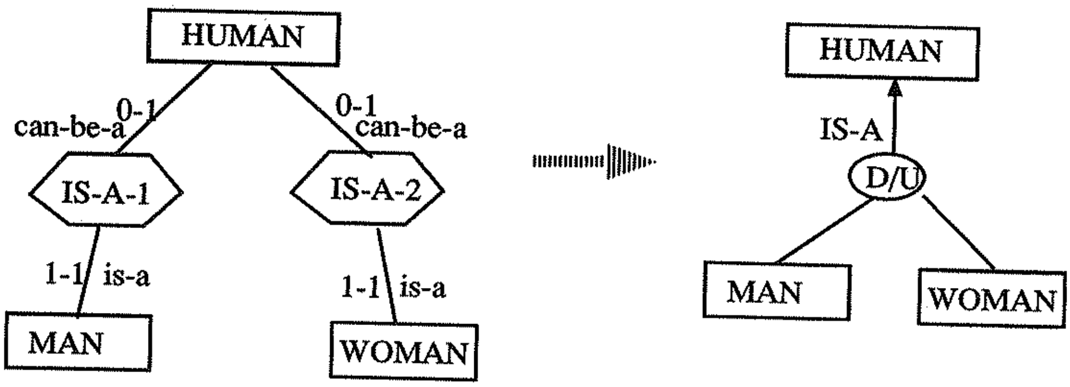
Goals Physical database design

Several relationship types into is-a link

Principles

An entity type linked to several entity types via very similar relationship types is specialized in these entity types and symetrically the specialized entity types are generalized in their commonly linked entity type.

Example



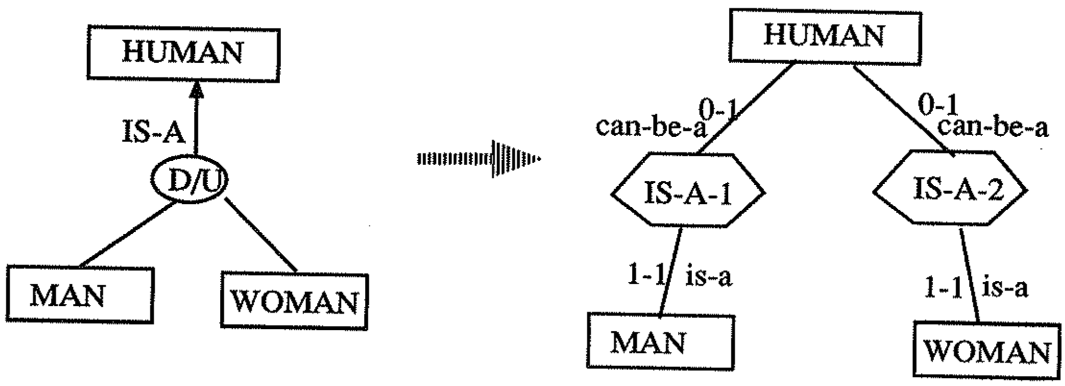
Goals conceptualization

Is-a link in several relationship types

Principles

The link IS-A between a specific entity types and a generic entity type is transformed in a relationship type without attribute.

Example



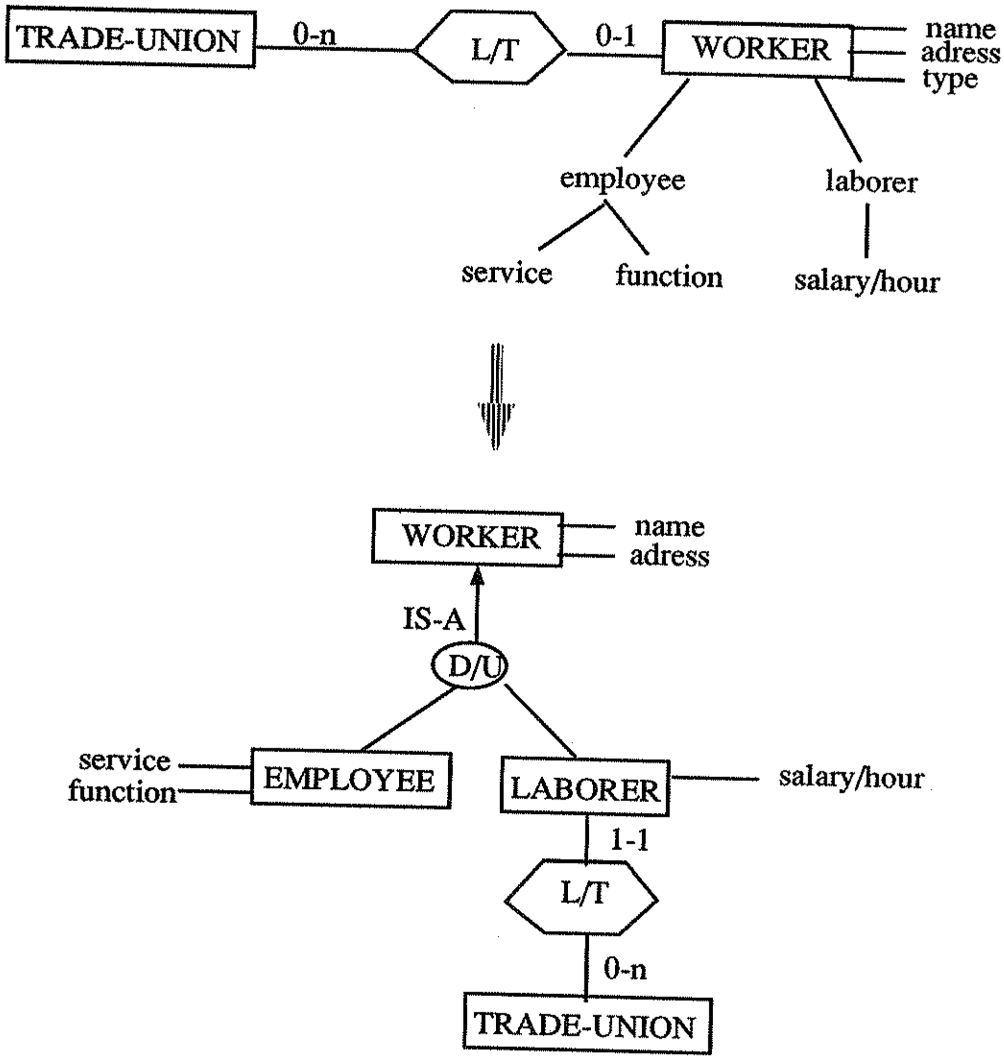
Goals Physical design

Entity-type into generic entity-type

Principles

An entity-type which modelized several classes of population is transformed in a generic entity type, each subclass is modelized in a specific entity type.

Example



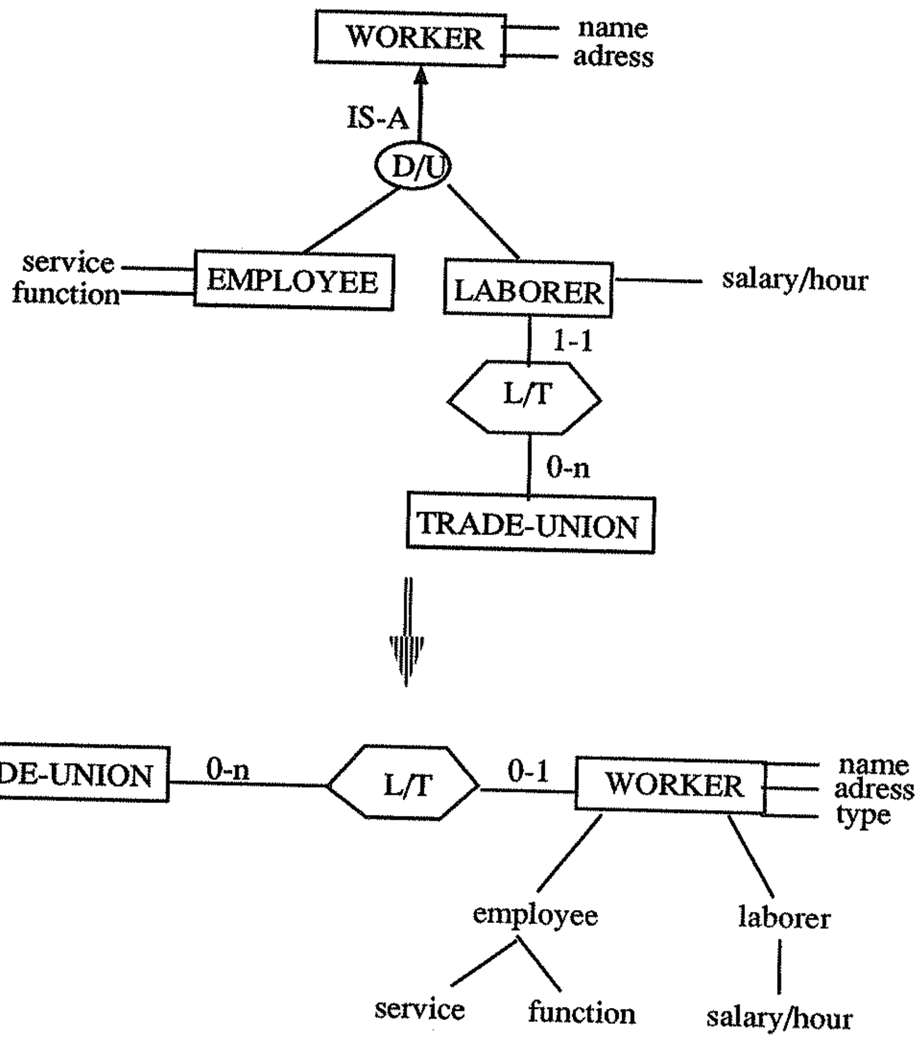
Goals conceptualization

Eliminate Is-a link keeping the generic entity-type only

Principles

The link IS-A between a specific entity types and a generic entity type is eliminated , the generic entity-type only is kept.

Example



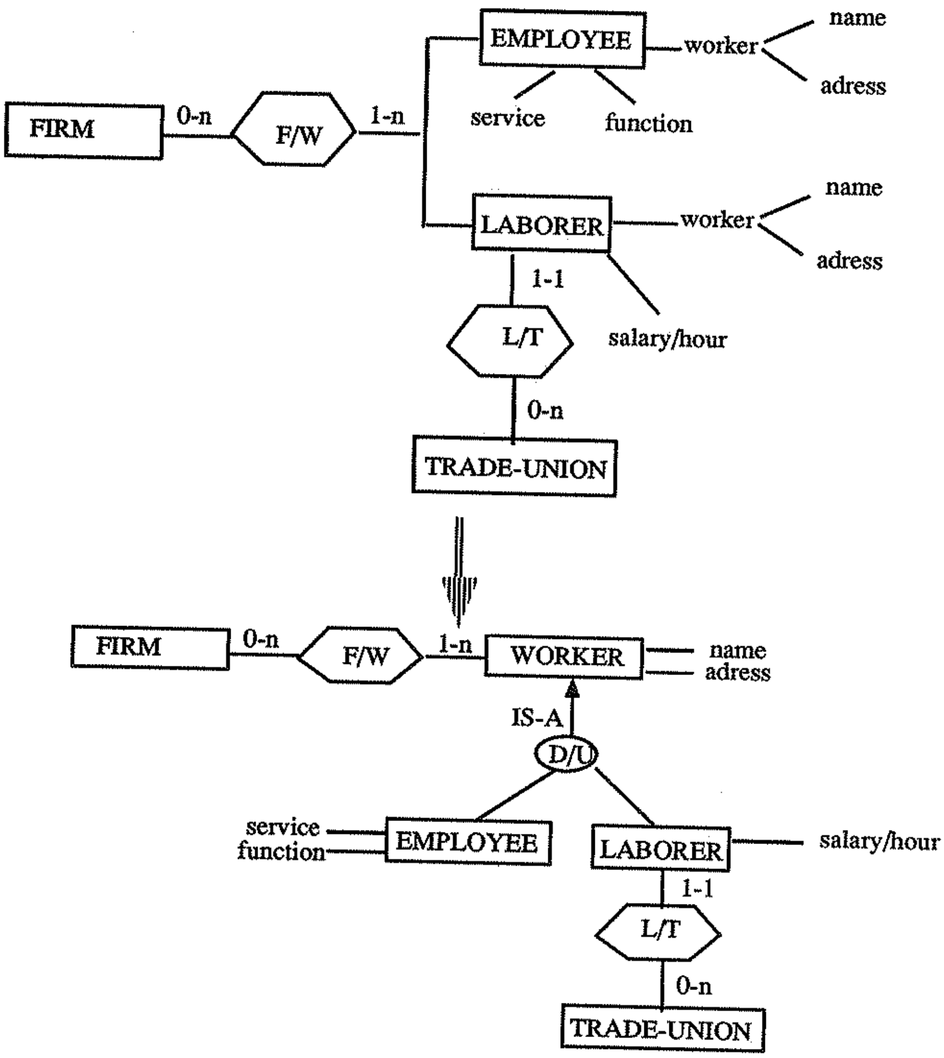
Goals **Physical design**

Several entity-types into several specifics entity-types

Principles

Some entity types having some common attributes , are transformed in specific entity types beeing a partition of a generic entity-type

Example

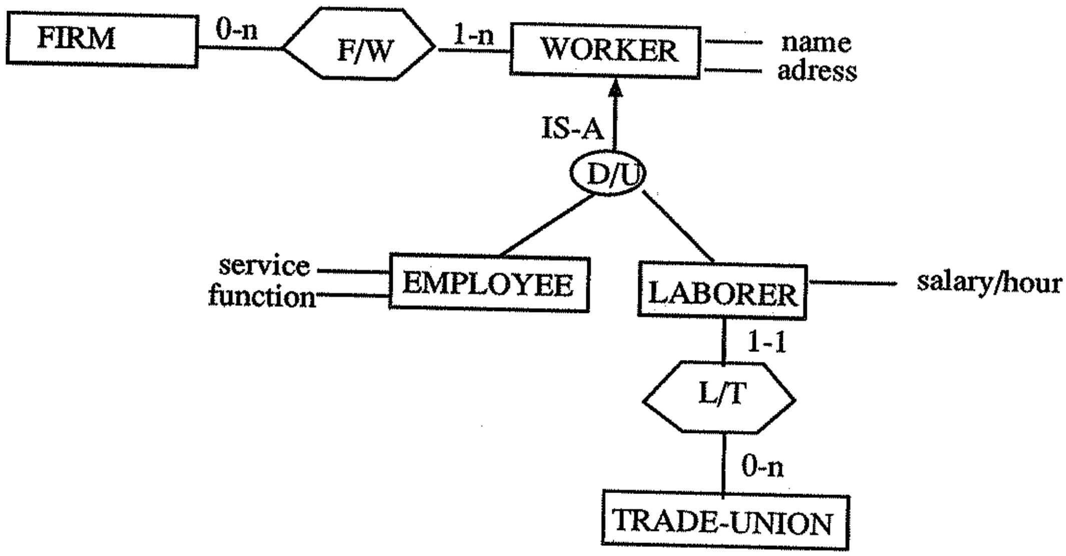


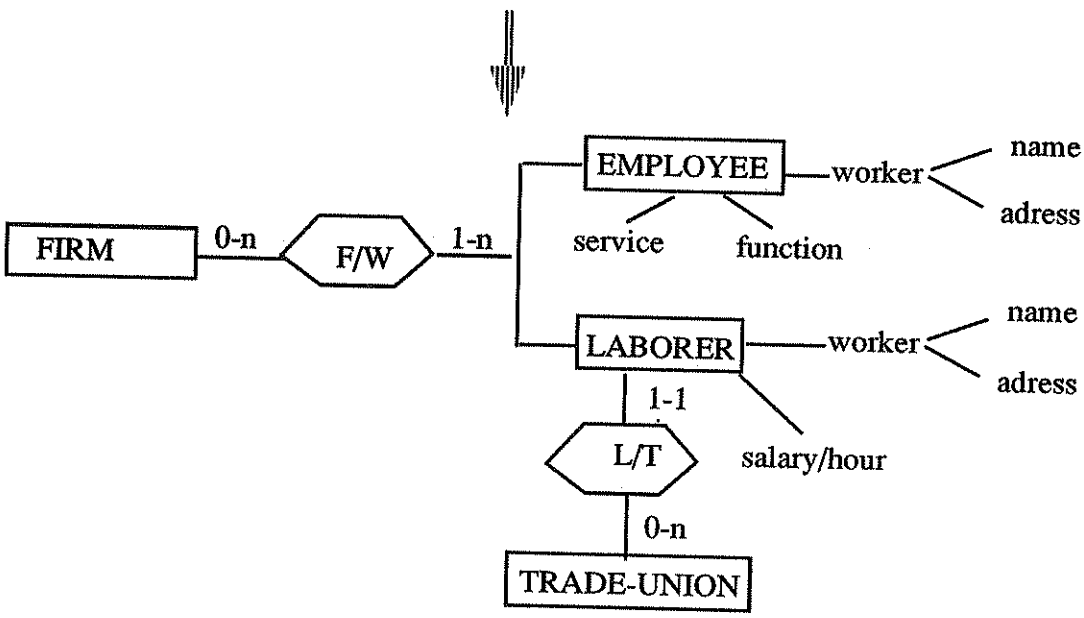
Eliminate is-a link keeping the specific entity types only

Principles

The link IS-A between specific entity types and a generic entity type is eliminated , the specific entity-types only are kept.

Example





Reverse Engineering Strategies and Methodology

Table of Contents

- Introduction
  - -- Strategy vs Methodology
  - -- Work Context
    Control of the RE Expert System
    Assistance for the RE'er
- RE Methodology
  - -- Overall Methodology
  - -- Methodological Assistance based on...
- What is a strategy?
  - -- A strategy is defined in terms of objectives
  - -- RE processes are means to reach objectives
  - -- Main Use ... to make suggestions to the reverse engineer
  - -- Strategies are based on the detection of significant patterns
- RE reasoning model
  - -- Conceptual view
    Current state
    Current Strategy
    Decision
    Action
  - -- 'Inference Engine' view

Introduction

Strategy vs methodology

- Methodology refers to methods that are techniques used to gain a result, according to an ordered working process
- Strategy implies ...
  decision points all through the RE process on the basis of the current state

Work Context

1. Control of the RE Expert System
   a. toolbox-oriented
   tools = RE processes + management processes
   ==> expert RE'er
   b. suggestion-oriented ==> novice RE'er
2. Assistance for the RE'er
   a. low
   handling of a great amount of data
   (toolbox-oriented ES, expert RE'er)
   b. medium
   'What must/can I do now?'
   (suggestive ES)
   c. high
   'Guide me in my RE activity'
   (suggestive ES, novice RE'er)

RE Methodology

Overall Methodology

- Main approach
  - -- starting with the (Cobol) source description(s)
  - -- towards a (conceptual) schema
- Use of standard inference reasoning techniques
  - -- Forward chaining (data driven)
    Analysis of existing data structures to determine applicable RE process(es)
  - -- Backward chaining (goal driven)
    Acquisition of specific data structures such as identifiers, relationship types, gen/spec structures,...
    Inter and intra schema integration
    Conformity

Methodological Assistance based on...

- mandatory steps
- a generic data model oriented to RE specific concepts

What is a Strategy?

1. **A strategy is defined in terms of objectives**

   a. main objective = overall RE objective
   'to rebuild the schema of an application'
   where 'schema' is ...
   - made up of several subschemas
   - at a specific level of abstraction
   - compliant with a specific data model, etc.

   b. splitted up into sub-objectives

   c. with various granularity levels
   - abstraction level --> methodology
   - data structure level

2. **RE processes are means to reach objective(s)**
   Transformation processes are means to reach model compliancy, 'better world adequacy',...
   Integration processes are means to reduce redundancy,...

3. **Main use ... to make suggestions to the reverse engineer**

4. **Strategies are based on the detection of significant patterns characterized by**

   a. specific data structures and/or other concepts

   b. violation of model compliancy rules

   c. redundant structures

   d. name relationships (synonyms, prefix, suffix,etc)

RE Reasoning Model

1. Conceptual view

RE reasoning totally defined by the following quadruplet

<current state [+ current strategy], decision, action>

where

- *current state* is (a part of) the schema under RE processing
- *current strategy*
- *decision* refers to the selection of one of the applicable RE processes according to the current state (and possibly the current strategy)
(= RE engineer's intention of action)
- *action* modifies the current state

1.1.Current State

a. ... is a state of the schema to be 'rebuilt' (design product)

b. a schema is considered as a set of objects

c. these objects are in a specific state ('completed', 'partially-extracted', 'user-defined',...)

1.2. Current Strategy

a. depends on the current state and previous actions ==> it is based on the detection of significant patterns and possible historical information

b. is used to make suggestions

1.3. Decision

a. various kinds of decision

- model-dependent

  Examples

  creation
  transformation (gen/spec)
  conformity checking

- methodology-dependent

  Example

  check for known losses of semantics implied by the forward engineering process

- strategy-dependent

  Example

  integrate two 'highly connected' schemas

b. dependencies may exist between decisions

Example

creation of an entity-type implies creation of an identifier

1.4. Action = Execution of a (RE) process

a. Changes the current state

b. May have precondition(s)

2. 'Inference Engine' view

- Step 1. Determine the set of RE processes to 'examine' according to the current strategy
(All RE processes if no strategy)

- Step 2. Among these ones, determine applicable RE processes according to the current state
Use preconditions of RE processes and recognition of structural patterns in schema.

- Step 3. Re-positioning in the strategy tree

Phenix Expert System

Components and Architecture

Table of Contents

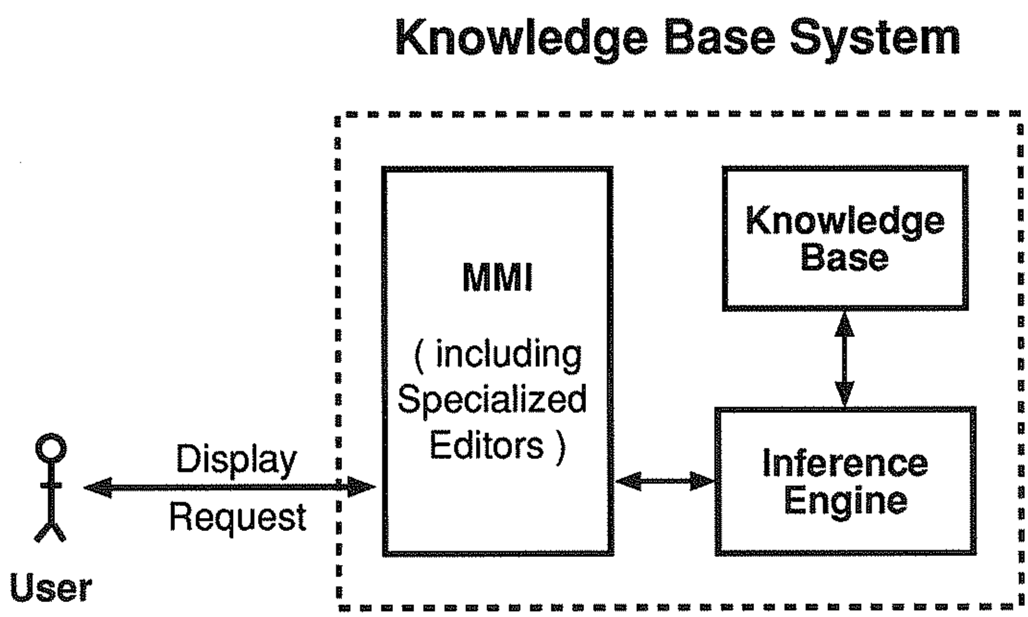
- Knowledge-Base System
  - -- Basic Components and Basic Architecture
  - -- Phenix Knowledge-Base System
- Blackboard Framework
  - -- Simple Blackboard Model
  - -- Blackboard Model with a Control Structure
  - -- Example. Finding Koalas
  - -- Phenix KBS and Control
- Development and Implementation Environment
  - -- Integrated Development Tools
  - -- Oriented towards ...
  - -- Main Features and Concepts
  - -- An Expert System in Smeci
  - -- Inference Engine Behavior
- Phenix ES Implementation with Smeci
  - -- Object Base
  - -- RE Process Base
  - -- Inference Engine
    Update of Smeci system-defined elements
    Control
    Strategy

Knowledge-Base System

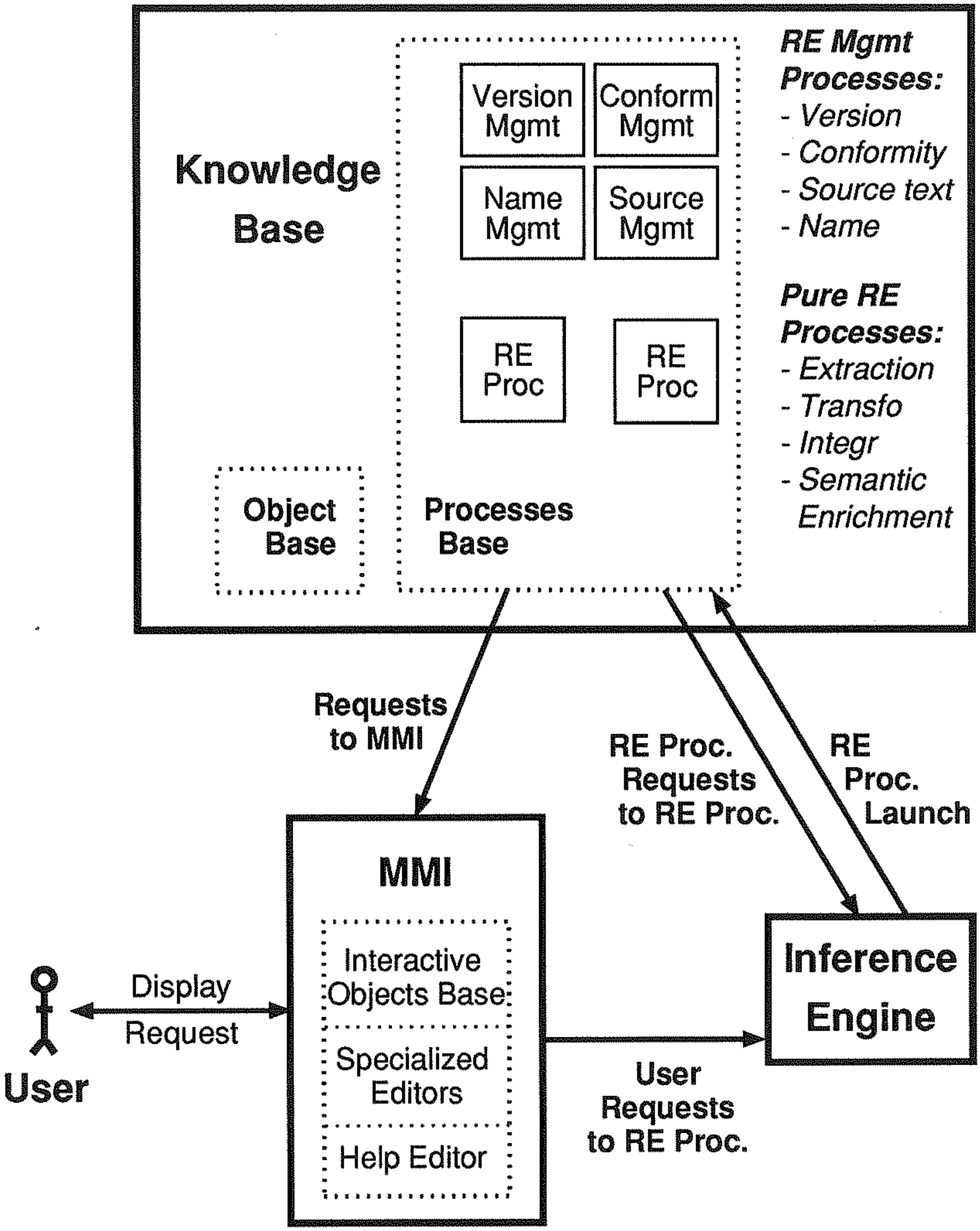
Basic Components

1. Knowledge Base
2. Inference Engine
3. Man-Machine Interface (MMI)

Basic Architecture

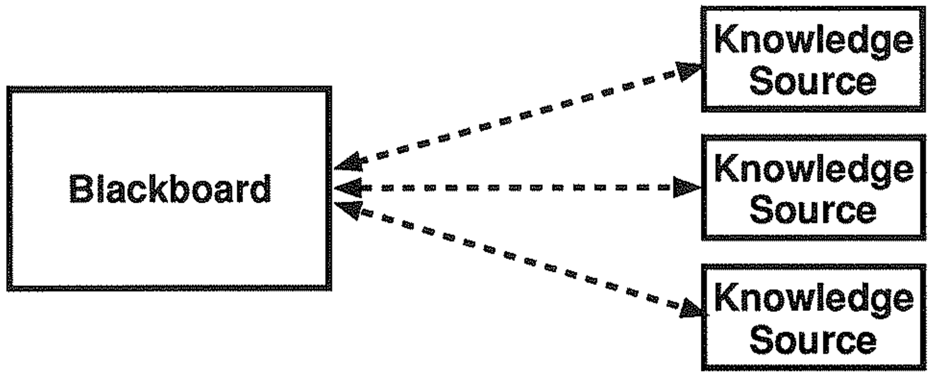


Phenix Knowledge-Base System



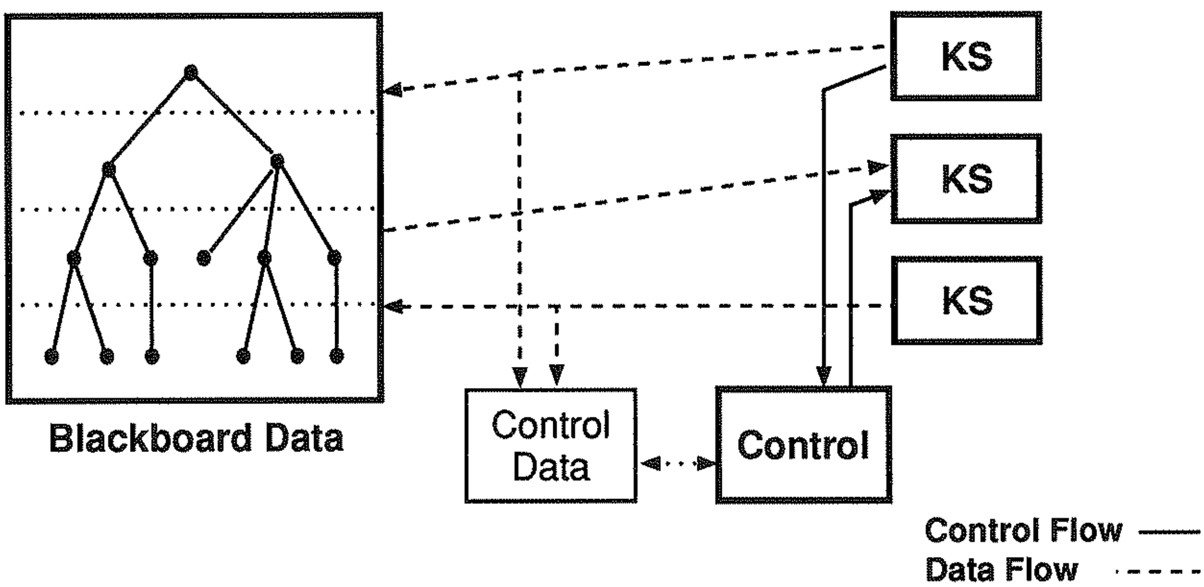
Blackboard Framework

1. Simple Blackboard Model



'Blackboards provides a means of coordinating the actions of multiple knowledge sources.'

2. Blackboard Model with a Control Structure

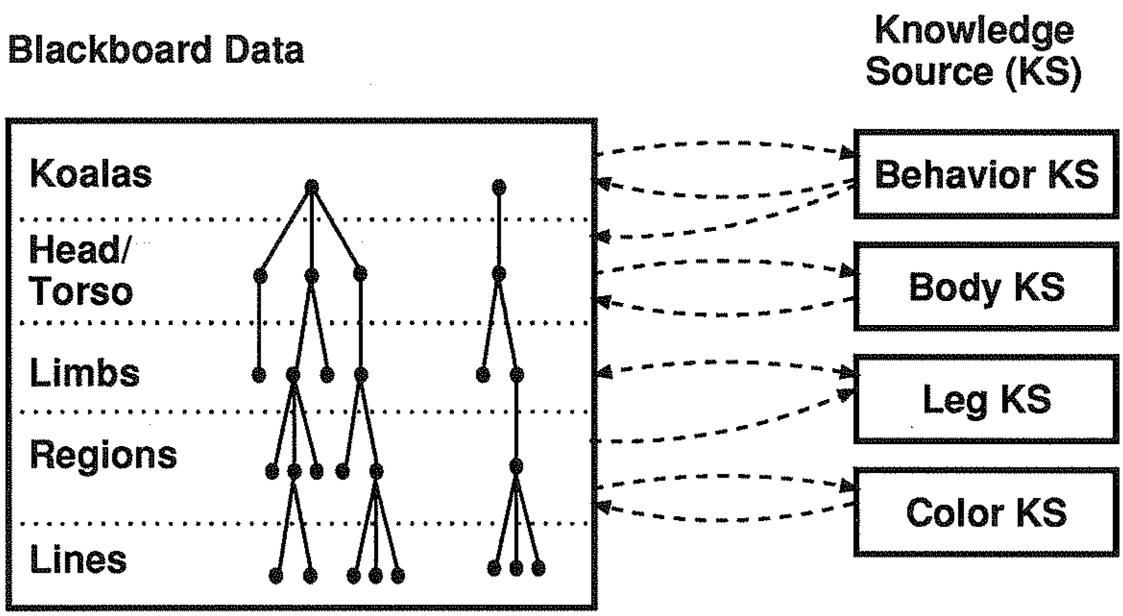


.. Blackboard ==
objects such as input data, partial solutions, alternatives, final solutions (and possibly control data)
.. Control selects the next (KS + Objects) to process

3. Example. Finding Koalas
(Feigenbaum, Penny Nii, 1974)

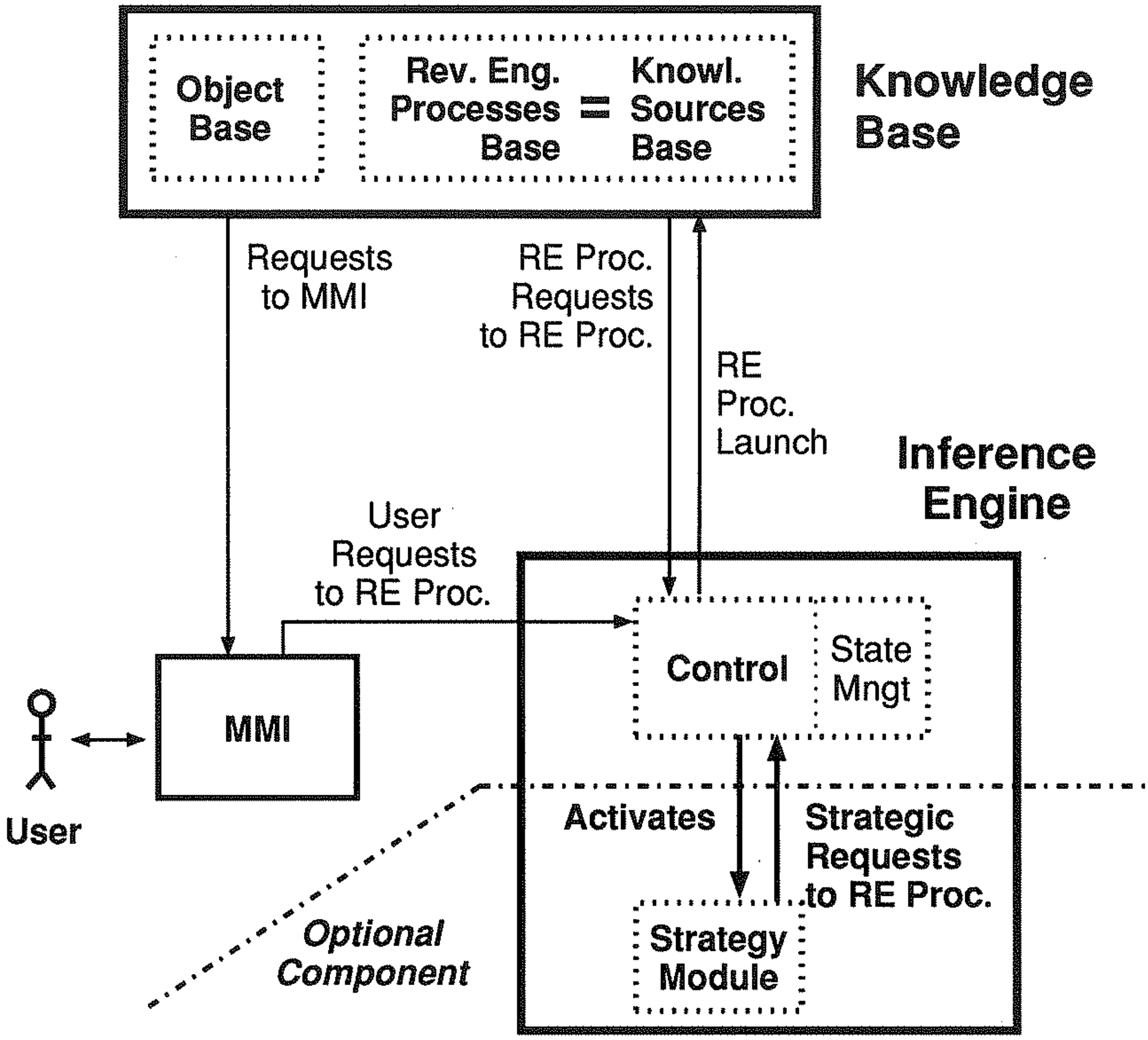
The problem: 'To find koalas in the forest of Austra lia'

Blackboard structure and knowledge sources



- description of the koalas in the scene as a part-of hierarchy
- no one source of knowledge can solve the problem
- specialist knowledge modules 'share' information about what they 'see' to help in the search for koa-las
- no operational framework

Phenix KBS and Control



Roles of the control module

- to find out the next process to launch (user request, RE request, strategy module)
- to manage and control 'states'

Development and Implementation Environment

Integrated Development Tools

- SMECI a general purpose ES shell
- AIDA/MASAI an automatic graphical interface generator
- based on LELISP

Oriented towards ...

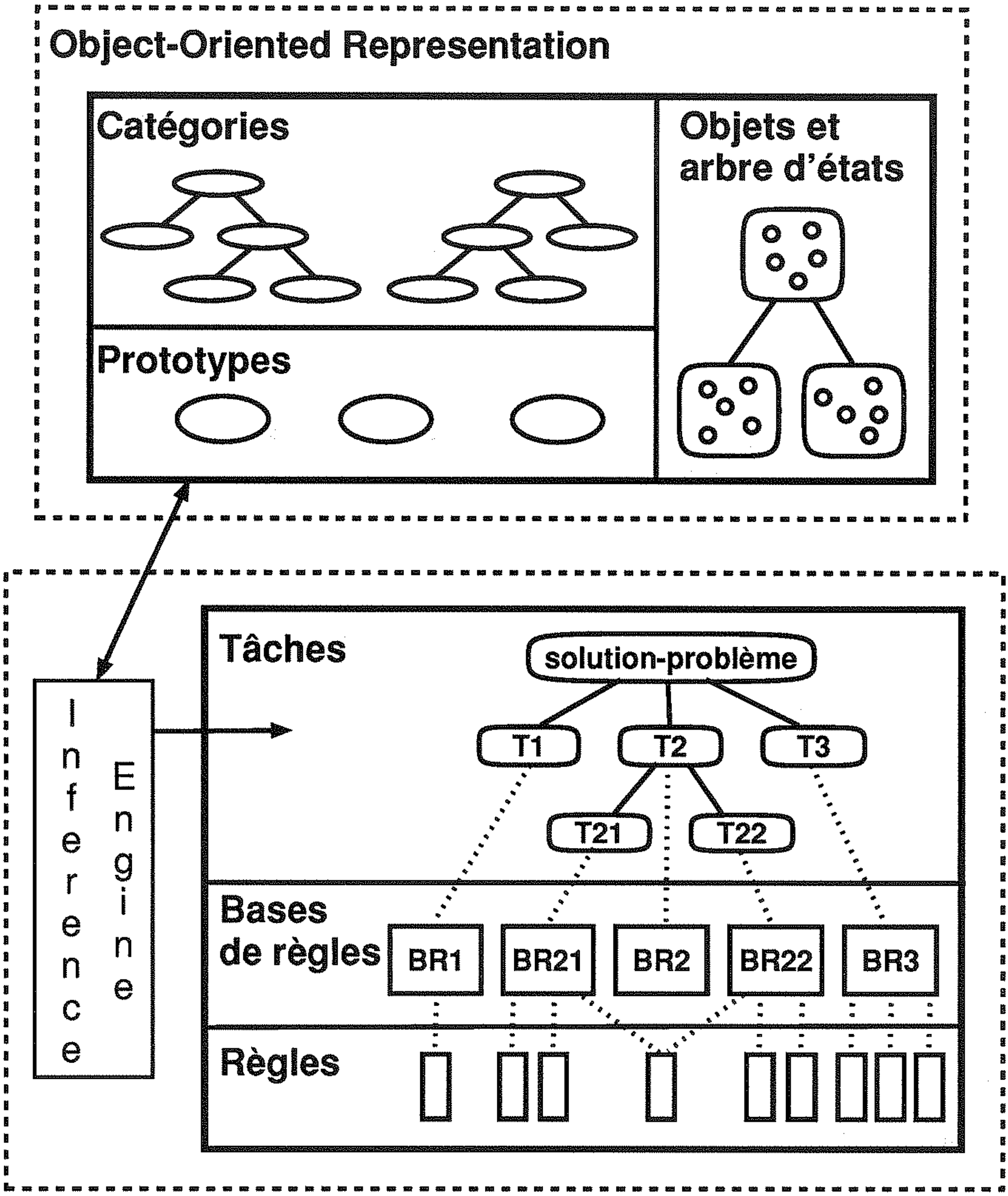
... the development of problem-solving expert systems

==> in the basic Smeci environment, no development framework is advised for a blackboard oriented expert system

Main Features and Concepts

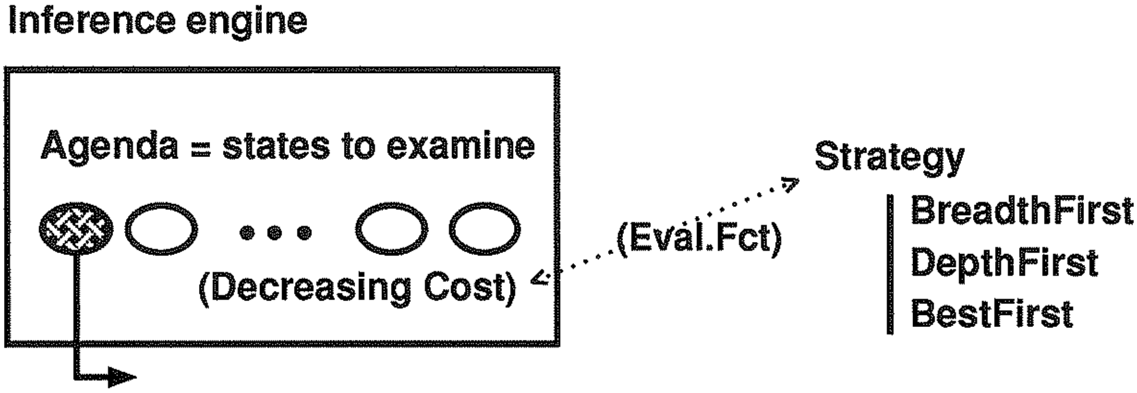
1. Knowledge Representation
   a. Concepts of the Application World
   --> Object-Oriented Representation
   b. Problem-Solving Knowledge
   --> 1st Order Rules structured in Tasks
   --> hierarchy of Tasks
   c. Procedural Knowledge
   --> methods, Lisp Functions, daemons
2. Inference Engine
   a. State = set of slot values of all existing objects
   (incl. system objects as task stack)
   b. State Tree
   rule application implies state(s) generation
   c. Agenda = list of states not expanded yet
3. Reasoning
   a. Strategy (DepthFirst, BreadthFirst, BestFirst)
   b. Control Parameters (instanciation-number, sorting-mode, firing-mode) used to define
   (1) which rule to execute among the applicable rules
   (2) when and how states are generated
4. Smeci <-> Aida/Masai Protocol

An Expert System in Smeci

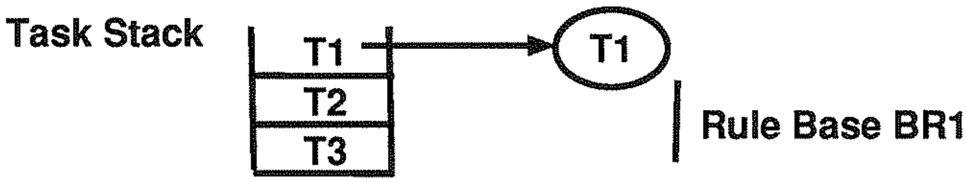


Inference Engine Behavior

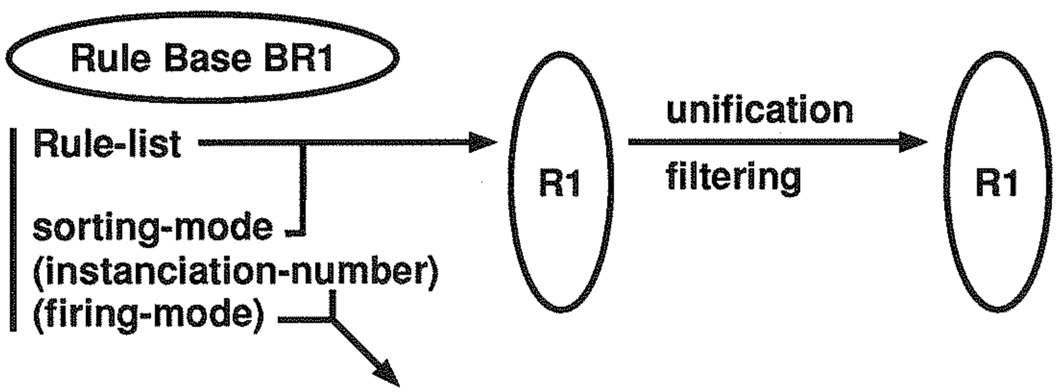
1. State Selection



2. Current Task Identification



3. Construction of the Set of Rules to Examine and Determination of Applicable Rules



4. 'Son' State(s) Generation according to Control Parameters of rule R1 and rule base BR1

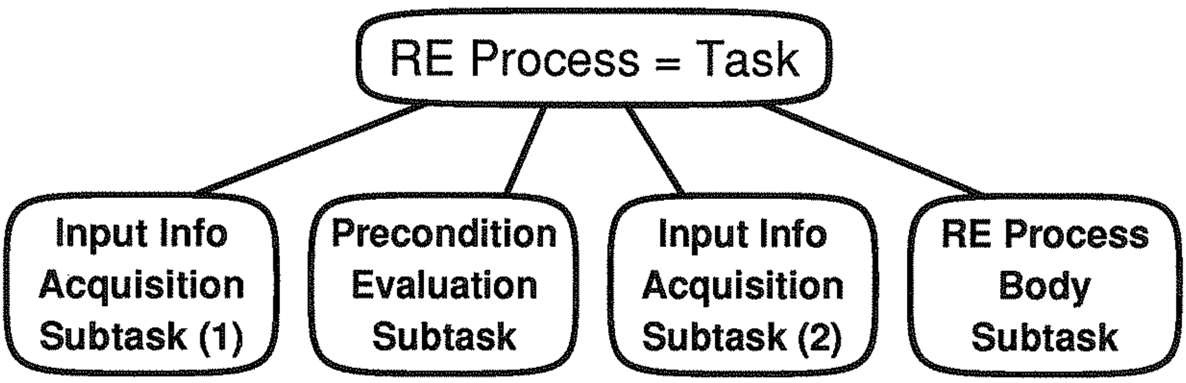
Phenix ES Implementation with Smeci

1. Object Base

'Smeci compliant' object base + functional interface

2. RE Process Base

RE processes are tasks
RE Process definition framework



- Input Information Acquisition Subtasks (1 and 2)

    input parameters and complementary information to acquire before body execution

- Precondition Evaluation Subtask

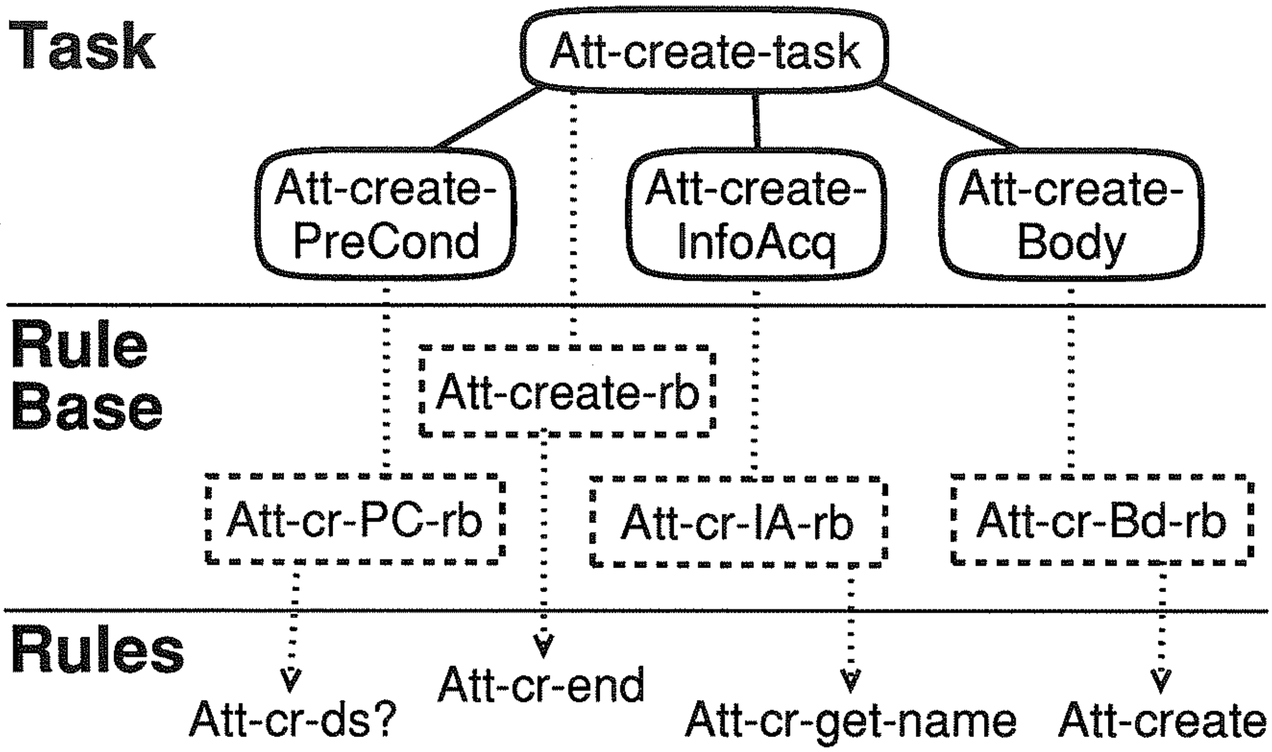
    conditions to be met to allow body execution

- RE Process Body Subtask

Example.

Semantic Enrichment - Creation of an attribute

Given a data-structure, this RE process creates a new attribute for it. A name for this attribute is optionally provided as input parameter.



- Att-cr-ds?
  if 1st-param is not a data struct. then send 'error' and stop
- Att-cr-get-name
  if not 2nd-param then build-a-name
- Att-create
  create an attribute for the data structure in 1st-param
- Att-cr-end
  send confirmation message

3. Inference Engine

3.1 Update of Smeci system-defined elements

- control of task stack via new control rules and tasks
  - -- pop-task / pop-rule
  - -- push-transition-task / push-transition-rule
  - -- pop-transition-task / pop-transition-rule

- redefinition of two system methods used to construct the set of rules to examine (step 3 of a cycle of the inference engine)
  ('sorting-mode' control parameter)
  - -- defmethod strictly-ordered (rulebase agenda)
  - -- defmethod ordered (rulebase agenda)

3.2 Control Module

- Basic Control Functions to ...
  - -- launch process (with input/output parameters passing)
  - -- evaluate precondition subtasks
  - -- manage result and error code
- Control Task(s) for ...
  - -- State Management
    Examples:
    'Undo' facility
    Multi-Hypothesis Evaluation
    ==> task implementation
  - -- Session Management
    Example:
    Session mode control and management process (toolbox, suggestion, automated)

3.3 Strategy Module

- Local Object Base for Specific Concepts
- RE Process
  - -- task implementation
  - -- 'launched' by the control module (Session Management Task)