Moving on Design

Design Strategies
Layering and Use of Packages

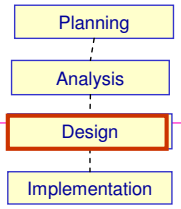
Lecture : 12
Date : 22-11-2005

Yannis Tzitzikas
University of Crete, Fall 2005

Outline

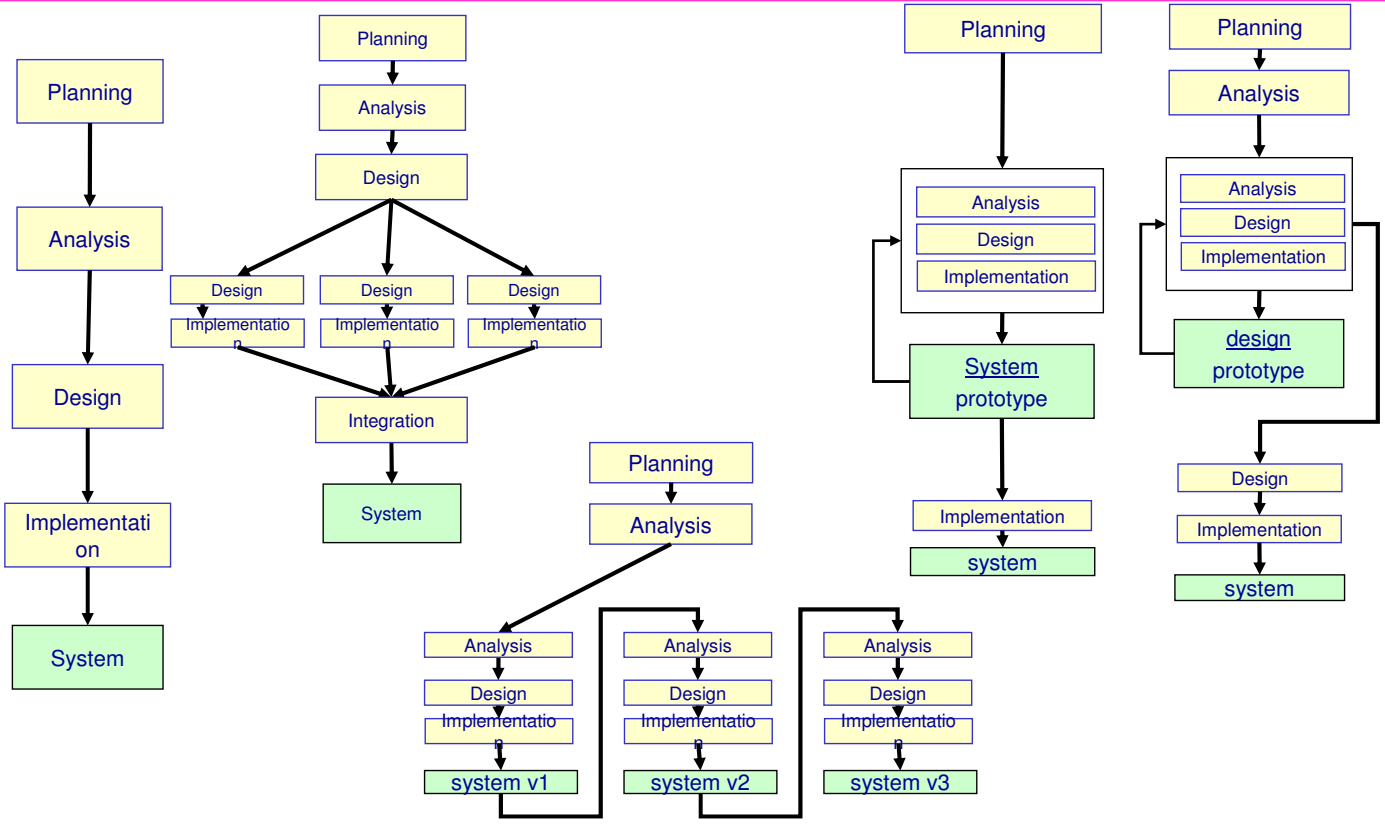
- The Design Phase (refresher)
- Design Strategies
  - Custom Development, Packaged Software, Outsourcing
  - Selecting a Design Strategy
- From Analysis to Design
  - From Analysis Models to Design Models
  - Partitioning the Problem - Layering
  - Packages and Package Diagrams
  - Packaging and Clustering

Refresher: The Design Phase

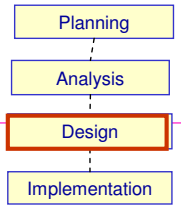


- How the system will operate in terms of
  - hardware, software and network infrastructure, the UI, forms and reports, databases, files, etc
- Steps:
  - 1. Design Strategy. Developed by company itself or outsourced to another firm, or buy an existing software package
  - 2. Architecture Design: hardware, software and network infrastructure, the UI, forms and reports, databases, files, etc
  - 3. Database and file specifications: define exactly what data will be stored and where
  - 4. Program design: programs that need to be written and what each will do.
- Outcome:
  - System Specification that is given to the programming team for implementation

Refresher: Design phase and Development Methodologies



The Design Phase>*Design Strategy*



- How the system will operate in terms of
  - hardware, software and network infrastructure, the UI, forms and reports, databases, files, etc
- Steps:
  - 1. Design Strategy. Developed by company itself or outsourced to another firm, or buy an existing software package
  - 2. Architecture Design: hardware, software and network infrastructure, the UI, forms and reports, databases, files, etc
  - 3. Database and file specifications: define exactly what data will be stored and where
  - 4. Program design: programs that need to be written and what each will do.
- Outcome:
  - System Specification that is given to the programming team for implementation

Design Strategies

Three Main Design Strategies

- Custom Development → Building the system from scratch
- Packaged Software → Buying a packaged software and customizing it
- Outsourcing → Hiring an external person/company to build the system

Ανάπτυξη κατά παραγγελία
Πακέτο Λογισμικού
Εξωτερική ανάθεση

Design Strategies
Custom Development

Custom development means building the system from scratch

Advantages
- Complete control (and freedom) over the functionality of the system
- Extensibility of the system (future needs may be taken into account)
- The skills of the staff improve (beneficial for future projects)

Weaknesses
- Takes time, requires hard work (the staff may be already overcommitted)
- Appropriately skilled staff is needed
- High risk

Design Strategies
Packaged Software

Buying a packaged system and customizing it
(commercial off-the-shelf (COTS) software product)

- many business needs are common to a large number of organizations
- there are thousands of commercially available programs
  - that support basic and ubiquitous needs (e.g. payrolls, orders, inventory, sales, etc)
- these programs range from small components to complete ERP (enterprise resource planning) systems
  - complete ERP systems have taken years and have cost millions of Euros to develop (SAP, PeopleSoft, Oracle, Baan)

Design Strategies
Packaged Software (II)

This decision strategy introduces an additional phase, the "procurement phase" which is a special decision analysis phase comprising the following steps:

1) Identify products that could fit the needs of the project
   - sources: special magazines, journals, Web sites, other organizations with similar needs
2) solicit, evaluate and rank vendor proposals
3) select and recommend the best of them
4) contract with the awarded vendor to obtain the product

Procurement phase: φάση προμήθειας

Design Strategies
Packaged Software (III)

2) solicit, evaluate and rank vendor proposals

Commonly, this step involves preparing an RFQ or an RFP document:

RFQ (Request for Quotations) (αίτηση προσφοράς)

used when you have selected the product but the product is available from various distributions. Objective: negotiate configuration, prices, maintenance contract.

RFP (Request for Proposals) (αίτηση προτάσεων)

used when there are several different vendors and products that are candidate. RFP is a superset of RFQ because the former describes in detail the requirements of the project (it is like the Requirements Specification Document)

Design Strategies
Packaged Software (IV)

Advantages

- It takes a short time to buy and install it (no extra staff is needed)
- Low risk (packaged software has already been written, tested and proved to be effective)

Weaknesses

- Not full control over the functionality of the system (we cannot customize everything)
  - sometimes the business processes of the organization have to adapt to the software!
- Extensibility of the system for the future needs of the organization is not ensured
- The skills of your staff do not improve
- The price of the packages software is sometimes high
- The organization depends on the vendor
  - maintenance (plus cost of maintenance)
  - Risk if the vendor bankrupts

Design Strategies
Outsourcing

Hiring an external person/company to build the system

Types of contracts:

- ***time and arrangements***
  - payment based on the time and the expenses
  - [-] the bill may climb high
- ***fixed-price contract***
  - requirements should be very clear
  - [-] usually outsourcers are not willing to accept changes in the requirements
- ***value-added contract***
  - the outsourcer will have some share of the benefits from the system
  - becomes more and more popular

Design Strategies
Outsourcing (II)

Advantages

- You have the full control over the requirements
- The external party may be more skilled than you

Weaknesses

- Not full control over the internal architecture of the system
- The skills of your staff do not improve
- You may have to share confidential information
- You need a person to manage the outsourcing
- Risk if you haven't specified well the requirements
  - never outsource something you don't understand
- Risk if the external party fails

Selecting a Design Strategy

Selecting a Design Strategy

Custom Development	**Packaged System**	**Outsourcing**
Unique business need	Very common business need	The business need is not crucial for the organization
Your staff has technical skills and you want to enhance them	Technical skills are lacking and it is not strategically important to build them	Technical skills are lacking and it is not strategically important to build them
Loose and flexible deadline	Short timeframe with strict deadline	A "good" outsourcer is available
	You want to avoid risk	

What about your project?

Selecting a Design Strategy
Matrix of Alternatives

We can use a matrix of alternatives (like the one we used in feasibility analysis) for organizing the pros and cons of each alternative. We could also employ a weighting scheme or any other decision theoretic method for selecting the best strategy for the project at hand

Description	Custom Development	Packaged Software	Outsourcing
Operational feasibility			
Technical feasibility			
Economic feasibility Cost Payback period ROI			
Schedule feasibility			

Plus whatever other criteria are appropriate

Hereafter we assume that we have selected the custom development design strategy

From Analysis Models to Design Models

From Analysis Models to Design Models

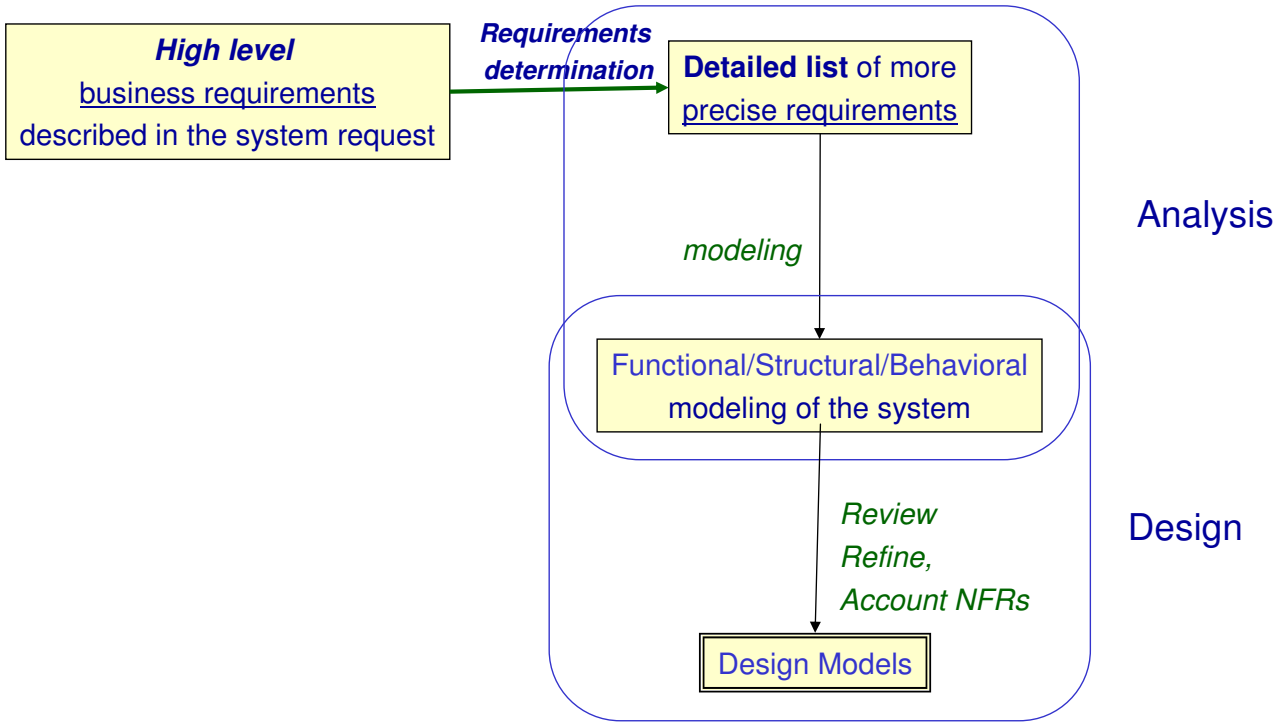
Purpose

- The analysis models we have created so far have mainly focused on FRs
- We should now take into account the **NFRs** too
- In OO: we have to review and refine the models we have created so far
  - by adding system environment details

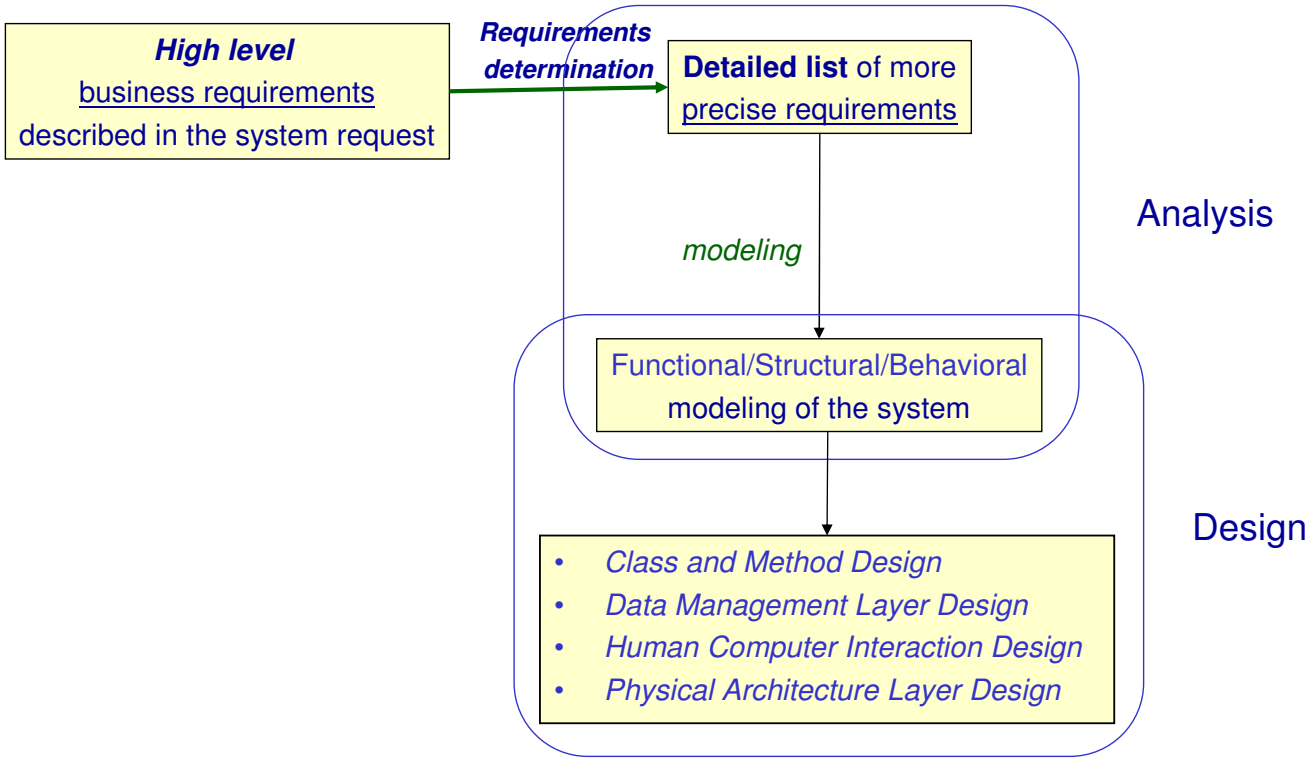
NonFunctional Requirements (NFR)

- Describe behavioral properties of the system, in terms of characteristics of the form:
  - performance, usability, security, legislative, privacy
- Describe how well the system should support the functional requirements
- We could also consider them as "*constraints*" that restrict the ways that we could use for implementing the FRs

From Analysis Models to Design Models



From Analysis Models to Design Models



From Analysis to Design
Technically, the Design Phase involves:

- *Class and Method Design*
- *Data Management Layer Design*
- *Human Computer Interaction Design*
- *Physical Architecture Layer Design*

- class specifications (detailed Class Diagrams)
- Contracts, Method Specifications

- file structure, database

- Use Scenarios, Windows Navigation Diagrams, Interface Standards, User Interface Templates

- Deployment Diagrams, Hardware/Software Specifications

Each of the above topics will be covered in the subsequent lectures

One of the oldest questions in software methods

How to break a large system ?

Reasons: *if big then hard to understand, extend, update*

Analysis Phase:

In the past: Functional Decomposition (used the days where process and data were separated)

today: Use Cases

Design Phase:

Identify layers and packages (packages are sometimes called subsystems)

Packages are vital tools for large projects

Packages are also good for testing (testing in package basis)

Identification of *Layers*

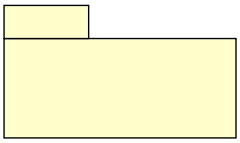
- So far we have focused on the **problem domain layer** (else called conceptual perspective)
- Now we have to add system environment information
- **One way to avoid overloading the designer is to use *layers***

An example set of layers:

Problem Domain	What we have seen so far
Data Management	Persistence of objects
Human Computer Interaction	User Interface classes
Physical Architecture	Communication, software architectures
Foundation Layer	e.g. fundamental data types (usually Included with the oo development environment)

layers can be represented using the higher level (UML) concept of PACKAGE

Packages and Package Diagrams



Divida et impera (divide and rule)

Package

A package is a grouping construct that allows us to take any construct of UML and group its elements into higher-level units.

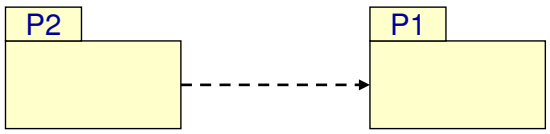
- It is commonly used to group classes

A package can group

- classes together into higher-level units
- design models, ..
- .....

Package Diagrams

shows packages and dependencies between them

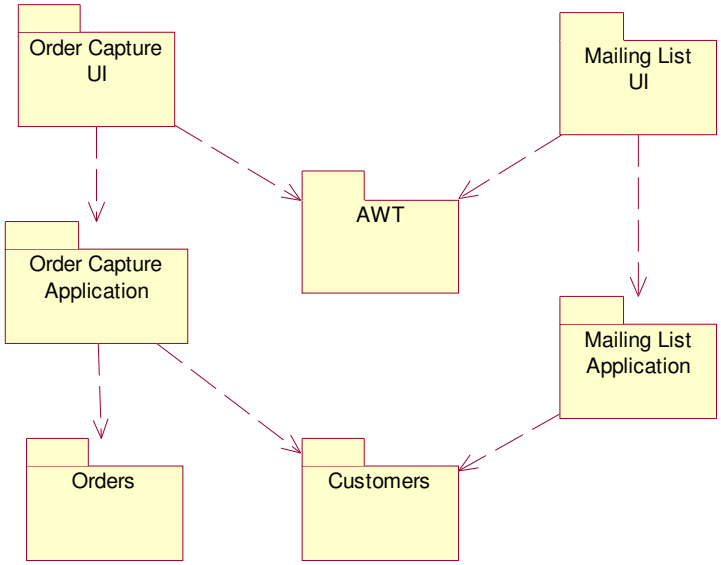


P2 depends on P1 if a change to P1 requires changing P1

Dependencies:

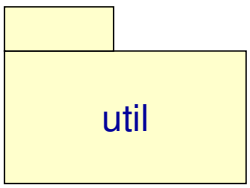
- Dependencies are not transitive
- A good practice is to remove dependency cycles (at least reduce them)
- The art of large-scale design: *minimize dependencies*

A package diagram (grouping classes)

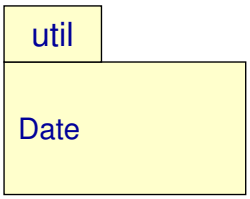


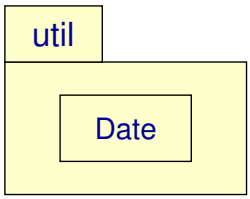
Ways of showing packages on diagrams

package name

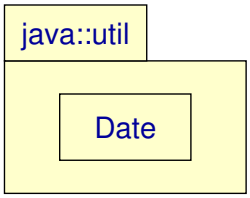


package name + contents

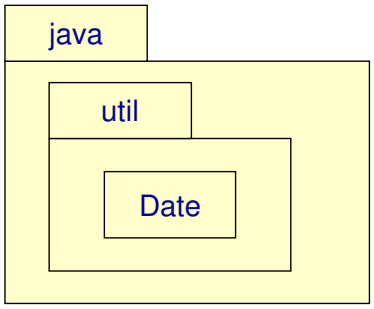




fully qualified package name

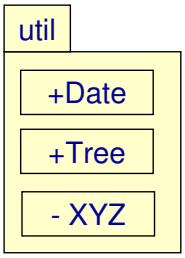


nested packages



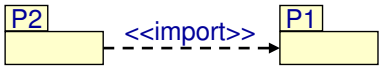
Packages, Visibility and Importing

- We can define the visibility of the elements owned by a package as we defined the visibility of the attributes and operations of a class.



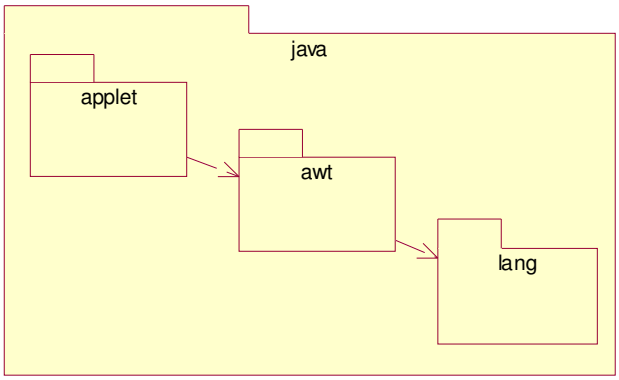
  - +: public
  - -: private: seen only the elements of the same package
  - #: protected: seen only by the elements of the same package and its children

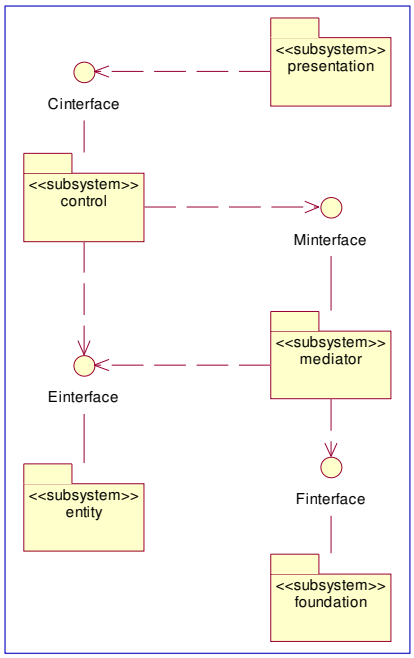
- To refer to the class of another package you need to use the qualified name of the class (package name::class name)
- ***import*** allows using direct names (not qualified ones)
- importing adds the public elements from the target package to the public namespace of the importing package. Notation in UML: <<import>>



"Import" vs "Access": "Access" is like "import" but you cannot reexport. So it is not transitive (import is transitive)

More Examples





Layers as Packages

Modeling Layers using Packages

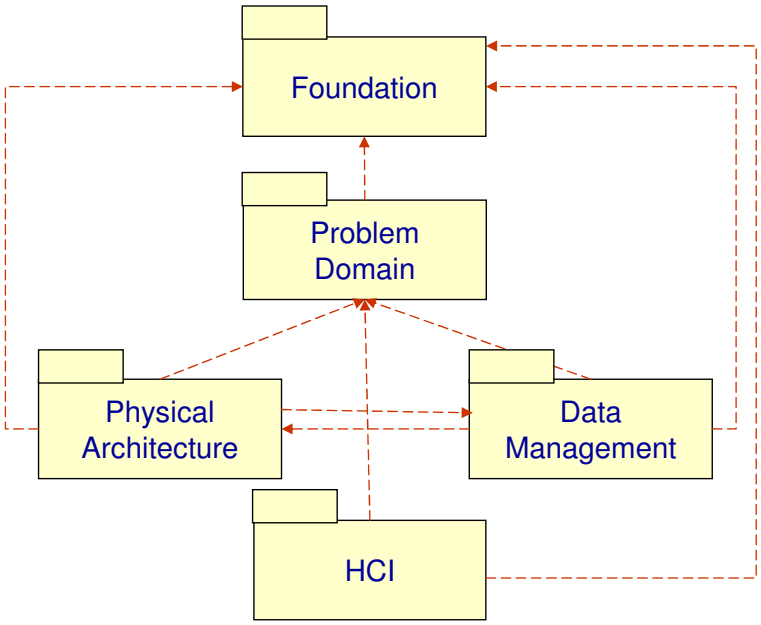
Problem Domain

Data Management

Human Computer Interaction

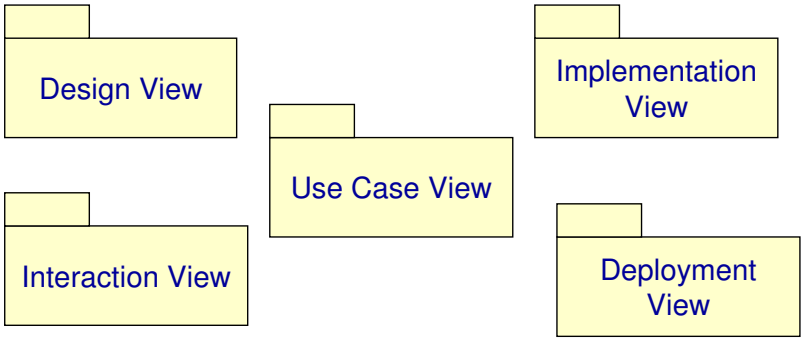
Physical Architecture

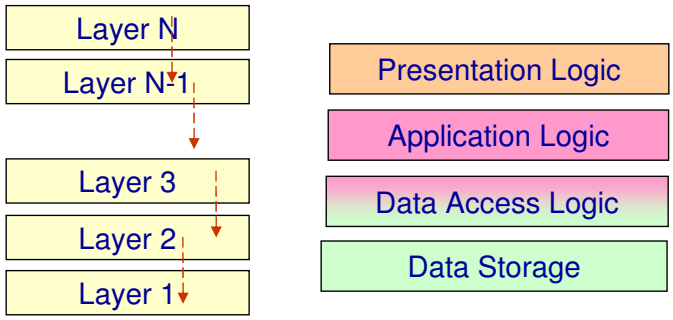
Foundation Layer



Packages for Modeling the various Architectural Views

We could view packages to model the views of an architecture

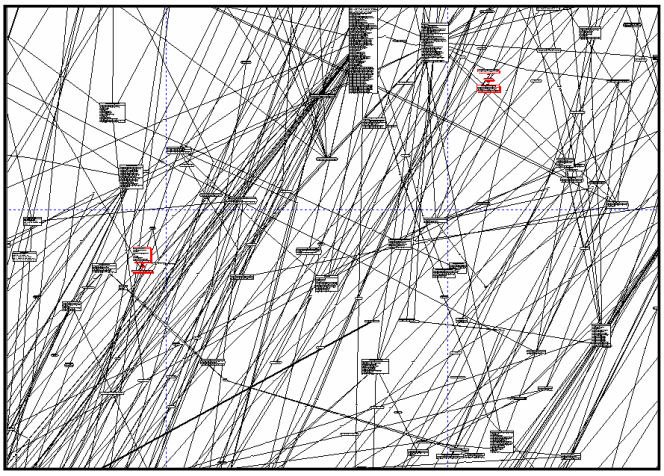


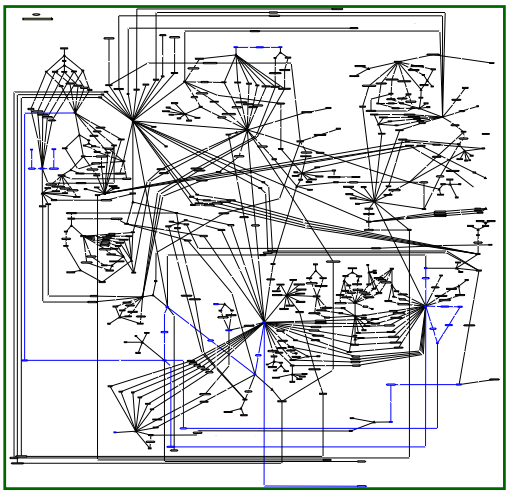


Packaging Classes

Packaging classes: Motivation

- Classes is the basic means for structuring an object-oriented system.
- *But is this structuring enough if we have hundreds of classes ?*





Tip: Use packages whenever a class diagram does not fit in an A4 paper

Packaging classes: Properties

Some properties :
- a class can be member of a single package
- a class should have a unique name within the package
- a package can contain both classes and packages

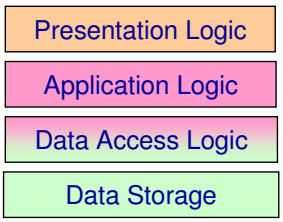
So we can define an hierarchic structure

Notes:
- UML packages correspond to packages in Java)

Packaging Classes

The packaging of classes can be done according to several different criteria, e.g.:

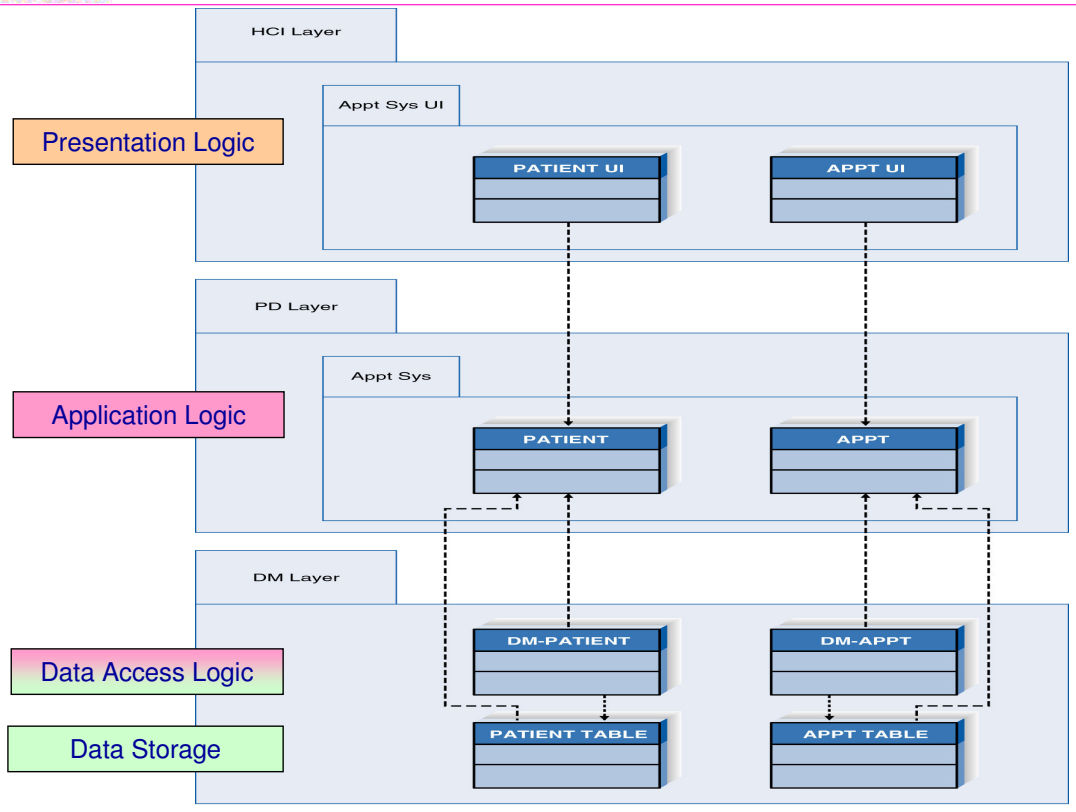
- by authorship (who wrote what)
- by functional relationship (e.g. Storage Manager, UI, MVC, …)
- by architectural tier, e.g. we could distinguish the following types of classes:



Other distinction:

- persistent database classes
- entity classes: those that will reside in the main memory
- boundary classes (interfaces)
- control classes: specify business logic function

Package Diagram of Appointment System



Dennis: SAD

Which classes to put in which packages ?

Two general principles:

- **Common Closure Principle**
  - the classes in a package should need changing for similar reasons
- **Common Reuse Principle**
  - the classes in a package should all be reused together (semantically close)

These principles drive us to the notion of ***cohesion***

Packaging classes: Hints and Tips

A well structured package

- is **cohesive** (providing a crispy boundary around a set of related elements)
- is **loosely coupled** (exporting only those elements other packages really need to see and importing only those elements necessary and sufficient for the elements in the package)
- is **not deeply nested** (human understanding is limited)

Cohesion =συνεκτικότητα/συνοχή, Cohesive = συνεκτικό
coupling = σύζευξη, loosely coupled = χαλαρά ζευγμένα

Packaging relates to clustering

Clustering

- ***Clustering*** is the process of grouping similar objects into naturally associated subclasses.
- This process results in a set of "clusters" which somehow describe the underlying objects at a more abstract or approximate level.
- The process of clustering is typically based on a "similarity measure" which allows the objects to be classified into separate natural groupings.
- A *cluster* is then simply a collection of objects that are grouped together because they collectively have a strong internal similarity based on such a measure.
- A *similarity measure* (or *dissimilarity measure*) quantifies the conceptual distance between two objects, that is, how alike or disalike a pair of objects are.
  - Determining exactly what type of similarity measure to use is typically a domain dependent problem.

Clustering

A clustering of a set N is a partition of N, i.e. a set $C_1,\ldots, C_k$ of subsets of N, such that:

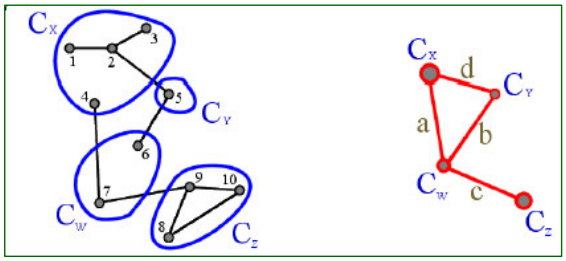
$$C_1 \cup \ldots \cup C_k = N \quad \text{and} \quad C_i \cap C_j = \varnothing, \text{ for all } i \neq j.$$

- Clustering is used in areas such as:
  - medicine, anthropology, economics, soil analysis, data mining
  - software engineering (reverse engineering, program comprehension, software maintenance)
  - information retrieval
- In general, any field of endeavor that necessitates the analysis and comprehension of large amounts of data may use clustering.

Packaging relates to graph clustering

Graph Clustering

- Graph clustering deals with the problem of clustering a graph
  - grouping similar nodes of a graph into a set of subgraphs

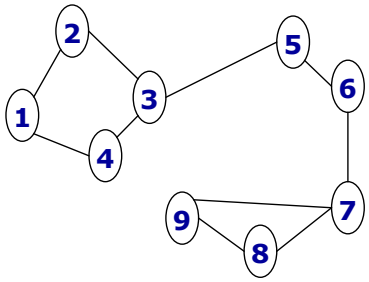


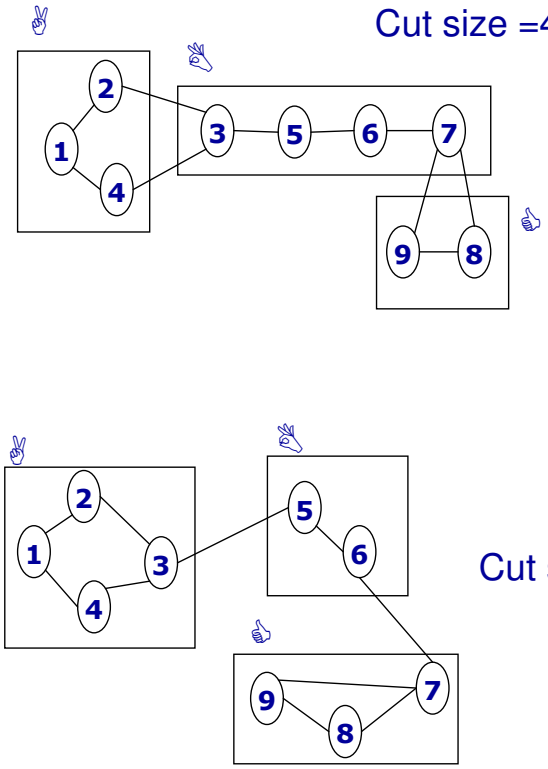
Quality criteria for graph clustering methods

Graph clustering methods should produce clusters with high cohesion and low coupling

- high cohesion:
  - there should be many internal edges
- low "cut size":
  - The cut size (else called *external cost)* of a clustering measures how many edges are external to all sub-graphs, that is, how many edges cross cluster boundaries.
- Uniformity of cluster size is also often desirable.
  - A uniform graph clustering is where $|C_i|$ is close to $|C_j|$ for all i,j in {1..k}

Example





Quality Measures for Graph Clustering

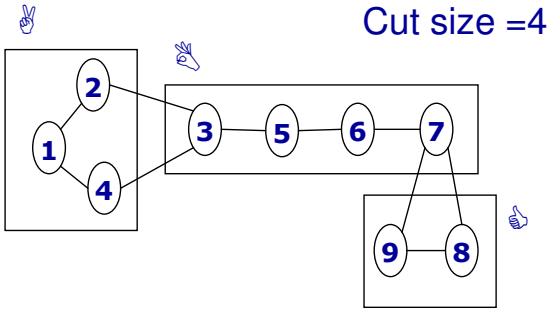
- There are several. One well known is the CC measure (Coupling-Cohesion measure)

$$CC = \frac{|E^{in}| - |E^{ex}|}{|E|}$$

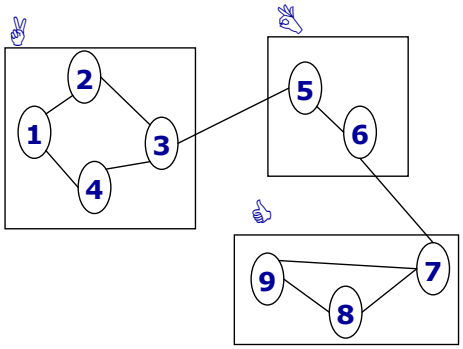
- E^{in}: the "internal" edges: those that connect nodes of the same cluster
- E^{ex}: the "external" edges: those that cross cluster boundaries
- maximum value of CC: 1
  - when all edges are internal
- minimum value of CC: -1
  - when all edges are external

Example

Cut size =4



$$CC = \frac{6-4}{10} = 0.2$$

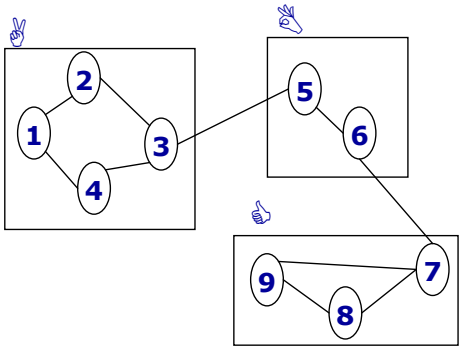


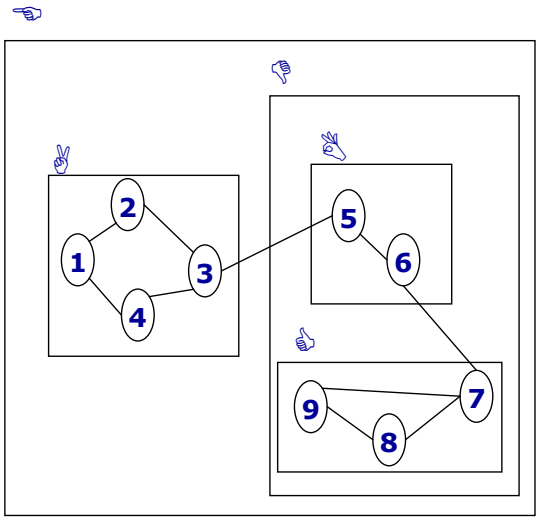
Cut size =2

$$CC = \frac{8-2}{10} = 0.6$$

Hierarchical Graph Clustering

- The clusters of the graph can be clustered themselves to form a higher level clustering, and so on.
- A hierarchical clustering is a collection of clusters where any two clusters are either disjoint **or** nested.





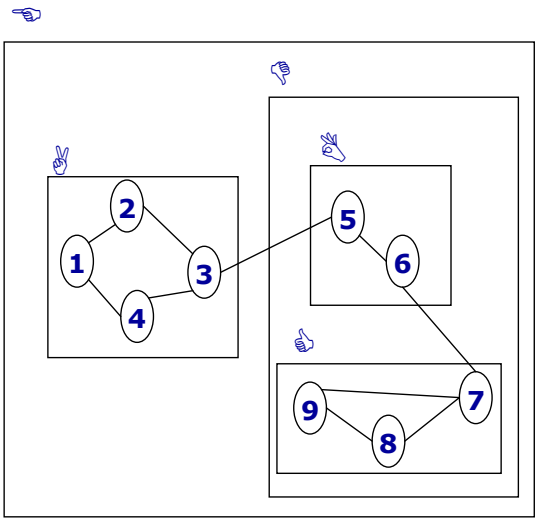
Hierarchical Clustered Graph

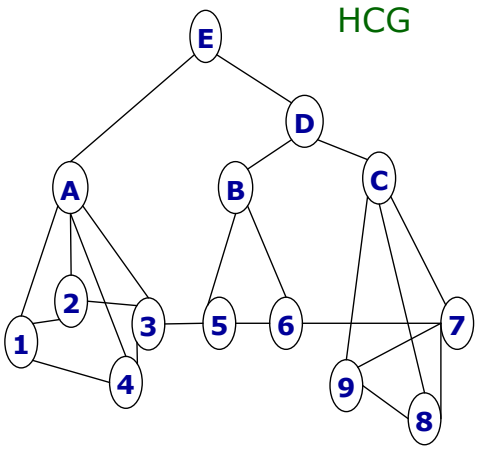
A Hierarchical Clustered Graph (HCG) is a pair (G,T) where

G is the underlying graph, and

T is a rooted tree such that the leaves of T are the nodes of G.

(the tree T represents an inclusion relationship: the leaves of T are nodes of G, the internal nodes of T represent a set of graph nodes, i.e. a cluster)



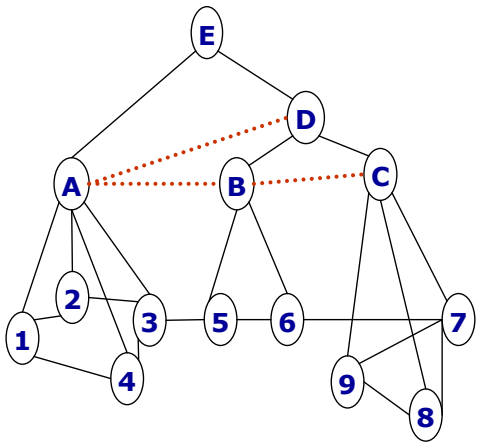


Implied Edges

Implied edges: edges between the internal nodes.

Two clusters are connected iff the nodes that they contain are related.

Multiple implied edges (between the same pair of clusters) can be ignored or summed up to form weighted implied edges. Thresholding can applied in order to filter out some implied edges



A Hierarchical Compound Graph is a triad (G,T, I) where (G,T) is a hierarchical clustered graph (HCG), and I the set of implied edges set.

Packages and Graph Clustering

Which is the graph G=(N,E) in our case?

- N: classes
- E: various dependency relationships
  - structural, e.g. associations
  - behavioral, e.g. the edges of communication diagrams
  - "semantical" (similarity of use/purpose: hard to infer automatically]

Which is the hierarchical compound graph (G,T, I) in our case?

- G (as before)
- T
  - leaves of T: nodes of N (i.e. classes)
  - internal nodes above leaves: packages
  - internal nodes of higher level: packages of packages
- I : the implied edges in our case are the dependency relationships between the packages

Packaging and Clustering: *Homework*

- Assume we have an implemented system whose classes are not packaged.
- *How could you partition them to packages?*

Algorithms for Graph Clustering

- Node similarity-based algorithms
- Kernighan-Li and variant algorithms
- Graph growing algorithms
- Spectral σ-section algorithms
- Multilevel partitioning algorithms
- ...

Summary

- Alternative strategies to design and create the new system: *Custom building, packaged software, outsourcing*
- *The object-oriented approach* to analysis and design is based on use-case modeling, is architecture centric, and supports functional, static and dynamic views of the system.
- When evolving analysis into design models, it is important to review the analysis models then add system environment information.
- *Packages and package diagrams* help provide structure and less complex views of the new system.
  - Set the context
  - Cluster classes together based on shared relationships
  - Model clustered classes as a package
  - Identify dependency relationships among packages
  - Place dependency relationships between packages

Reading and References

- **UML Distilled: A Brief Guide to the Standard Object Modeling Language** (3rd Edition) by Martin Fowler, Addison Wesley, 2004.
- **The Unified Modeling Language User Guide** (2nd edition) by G. Booch, J. Rumbaugh, I. Jacobson, Addison Wesley, 2004
- **Systems Analysis and Design with UML Version 2.0** (2nd edition) by A. Dennis, B. Haley Wixom, D. Tegarden, Wiley, 2005. CHAPTER 9
- **System Analysis and Design Methods** (6th edition) by Jeffrey L. Whitten, Lonnie D. Bentley and Kevin Dittman, McGraw-Hill, 2004, Chapter 12

How to break a large system ?

if big then hard to understand, extend, update

How to break an existing system in order to understand it?

A relevant method is presented next.

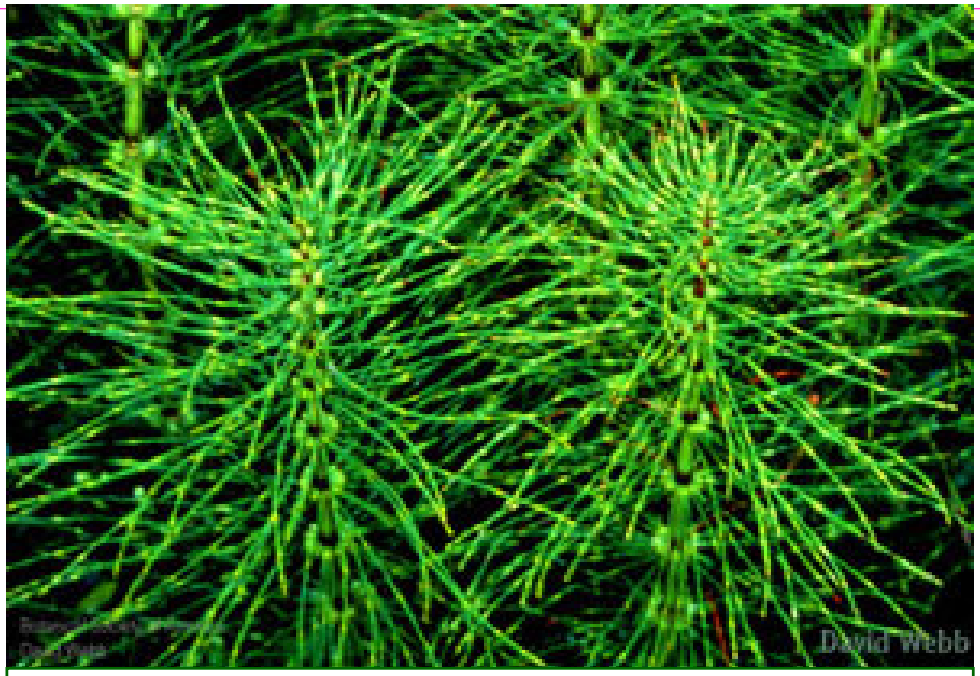
How to Tame a Very Large ER Diagram
(using Link Analysis and Force-Directed Placement Algorithms)

Yannis Tzitzikas[1] and Jean-Luc Hainaut[2]

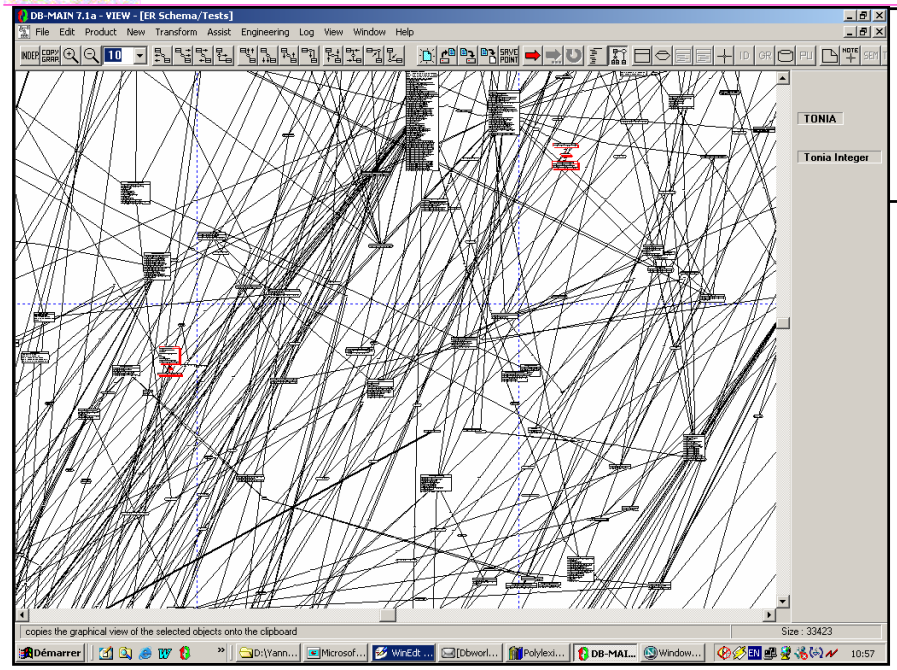
[1]University of **Crete** & **FORTH-ICS**

[2]University of **Namur** (F.U.N.D.P.), Belgium

*24th Intern. Conf. on Conceptual Modeling, **ER'2005**, Klagenfurt, Austria, Oct. 2005*



How to Tame a Very Large ER Diagram



Very small excerpt of a schema (DICO1) with **456** entity and **812** relationship types

Conceptual Diagrams

- They are useful and widely used in
  - Requirements Engineering
  - Reverse Engineering
  - Knowledge Representation
- Examples
  - ER diagrams
  - UML class diagrams
  - SWeb ontologies
  - ...

But why we use them ?

They aid communication

- customers vs analysts
- analysts vs domain experts
- analysts vs designers
- designers vs designers
- designers vs developers
- designers vs end users
- ...

But why we use them ?

They aid communication

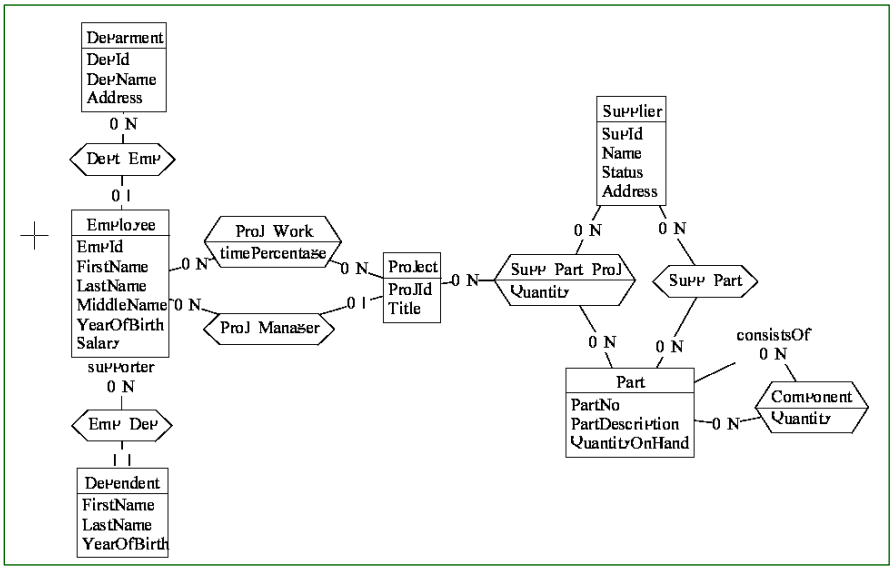
- customers vs analysts
- analysts vs domain experts
- analysts vs designers
- designers vs designers
- designers vs developers
- designers vs end users
- ...

```
create table Component (
    C_P_ID_Par char(10) not null,
    ID_Par char(10) not null,
    Quantity char(1) not null,
    constraint ID_Component primary key
        (C_P_ID_Par, ID_Par));
create table Deparment (
    ID_Dep char(10) not null,
    DepId char(1) not null,
    DepName char(1) not null,
    Address char(1) not null,
    constraint ID primary key (ID_Dep));
create table Dependent (
    FirstName char(1) not null,
    LastName char(1) not null,
    YearOfBirth char(1) not null,
    Supporter char(10) not null);
create table Employee (
    ID_Emp char(10) not null,
    EmpId char(1) not null,
    FirstName char(1) not null,
    LastName char(1) not null,
    MiddleName char(1) not null,
    YearOfBirth char(1) not null,
    Salary char(1) not null,
    ID_Dep char(10),
    constraint ID primary key (ID_Emp));
create table Part (
    ID_Par char(10) not null,
    PartNo char(1) not null,
    PartDescription char(1) not null,
    QuantityOnHand char(1) not null,
    constraint ID primary key (ID_Par));
create table Project (
    ID_Pro char(10) not null,
    ProjId char(1) not null,
    Title char(1) not null,
    ID_Emp char(10),
    constraint ID primary key (ID_Pro));
create table Proj_Work (
    ID_Emp char(10) not null,
    ID_Pro char(10) not null,
    timePercentage char(1) not null,
    constraint ID_Proj_Work primary key
        (ID_Pro, ID_Emp));
create table Supplier (
    ID_Sup char(10) not null,
    SupId char(1) not null,
    Name char(1) not null,
    Status char(1) not null,
    Address char(1) not null,
    constraint ID primary key (ID_Sup));
create table Supp_Part (
    ID_Par char(10) not null,
    ID_Sup char(10) not null,
    constraint ID_Supp_Part primary key
        (ID_Par, ID_Sup));
create table Supp_Part_Proj (
    ID_Par char(10) not null,
    ID_Pro char(10) not null,
    ID_Sup char(10) not null,
    Quantity char(1) not null,
    constraint ID_Supp_Part_Proj primary key
        (ID_Par, ID_Sup, ID_Pro));
-- Constraints Section
-- ___________________
alter table Component add constraint FKconsistsOf
    foreign key (ID_Par)
    references Part;
alter table Component add constraint FKCom_Par
    foreign key (C_P_ID_Par)
    references Part;
alter table Dependent add constraint FKEmp_Dep
    foreign key (Supporter)
    references Employee;
alter table Employee add constraint FKDept_Emp
    foreign key (ID_Dep)
    references Deparment;
alter table Project add constraint FKProj_Manager
    foreign key (ID_Emp)
    references Employee;
alter table Proj_Work add constraint FKPro_Pro
    foreign key (ID_Pro)
    references Project;
alter table Proj_Work add constraint FKPro_Emp
    foreign key (ID_Emp)
    references Employee;
alter table Supp_Part add constraint FKSup_Sup_1
    foreign key (ID_Sup)
    references Supplier;
alter table Supp_Part add constraint FKSup_Par_1
    foreign key (ID_Par)
    references Part;
alter table Supp_Part_Proj add constraint FKSup_Sup
    foreign key (ID_Sup)
    references Supplier;
alter table Supp_Part_Proj add constraint FKSup_Pro
    foreign key (ID_Pro)
    references Project;
alter table Supp_Part_Proj add constraint FKSup_Par
    foreign key (ID_Par)
    references Part;
```

But why we use them ?

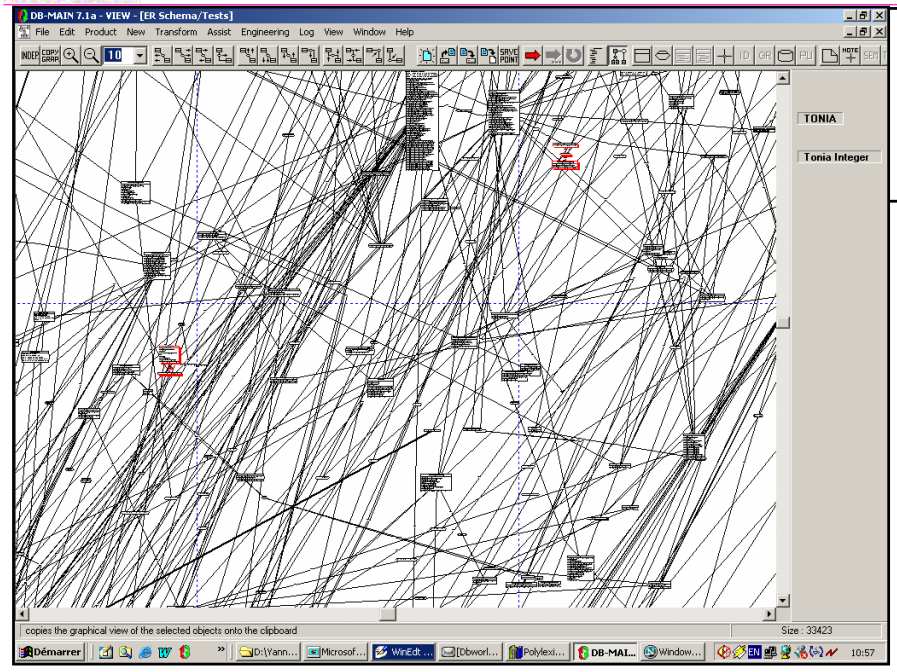
They aid communication

- customers vs analysts
- analysts vs domain experts
- analysts vs designers
- designers vs designers
- designers vs developers
- designers vs end users
- ...



A picture is worth a thousands of words

However, their usefulness **degrades rapidly** as they grow in size



Very small excerpt of a schema (DICO1) with
456 entity and
812 relationship types

Big schemas become more and more frequent

- SAP database consists of about 30.000 tables
- Semantic Web applications
- **Reverse engineering**

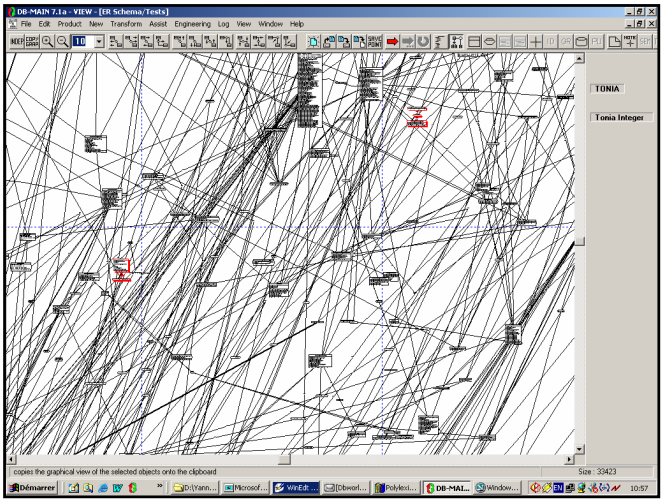
A conceptual diagram with thousands of elements does not worth a lot.

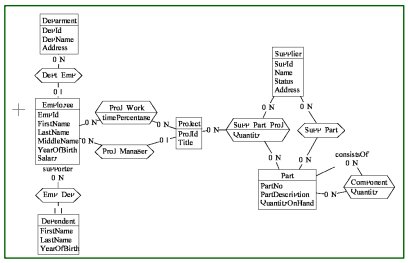
(unless it has a a regular form)

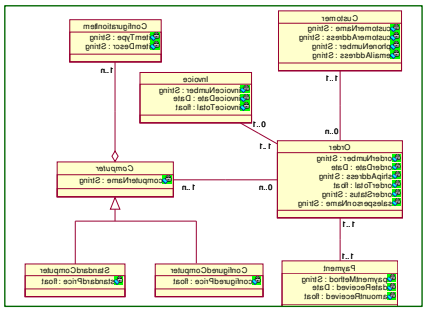
The objective:

Device techniques
to aid the understanding and the visualization
of large conceptual diagrams

E.g. generation of more abstract
or focused views







Existing Solutions for Managing Large Diagrams (1/3)
[1] ER Clustering

- The classical methods require human input
  - [Felldman&Miller'86,Teory et al.'89, Gandhi et al.94, Campbell et al.'96]
- Automatic methods not tested in large ER diagrams
  - [Akoka&Comyn-Wattiau'96, Raugh&Stickel'92]

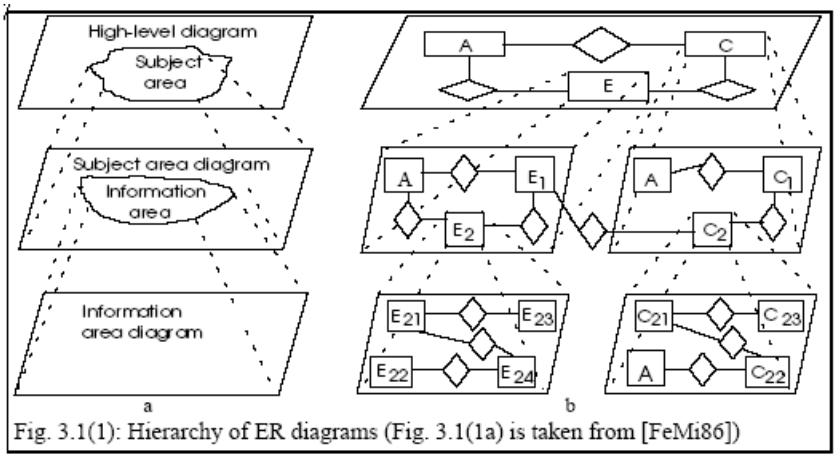
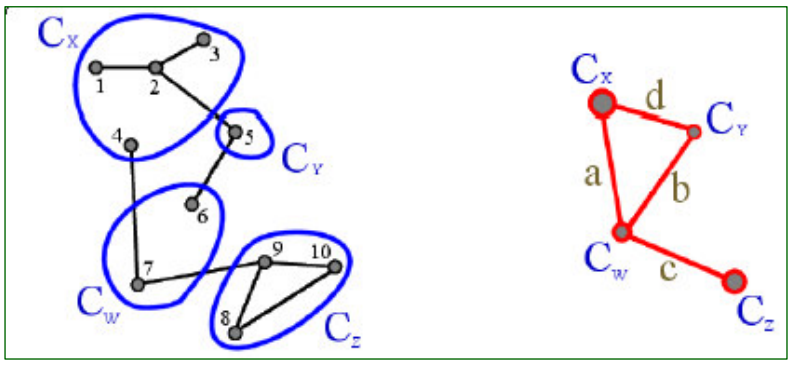


Fig. 3.1(1): Hierarchy of ER diagrams (Fig. 3.1(1a) is taken from [FeMi86])

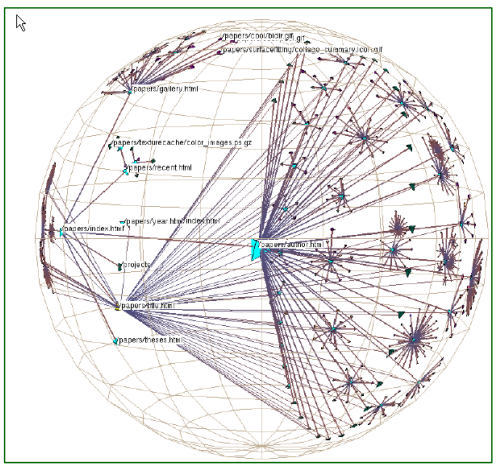
Existing Solutions for Managing Large Diagrams (2/3)
[2] Graph Drawing and Abstraction

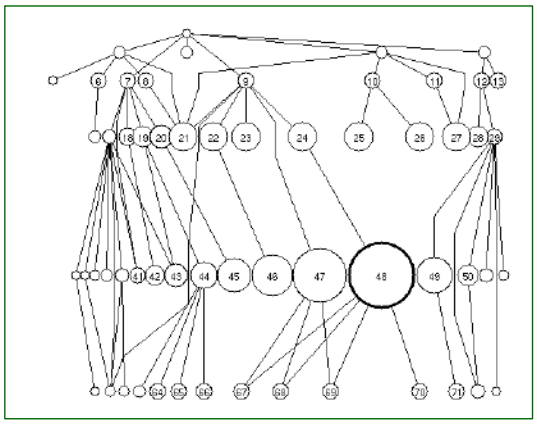
– Automatically generated layouts are not satisfying
  • Most (if not all) of the layouts are still created manually
– **Hierarchical decomposition techniques** that are used for visualizing big plain graphs have not been applied or tested on conceptual diagrams



Existing Solutions for Managing Large Diagrams (3/3)
[3] Visualization methods

– Not very helpful for our problem
  • Hyperbolic trees: appropriate only for trees
  • FishEye View: not really helpful and computationally hard



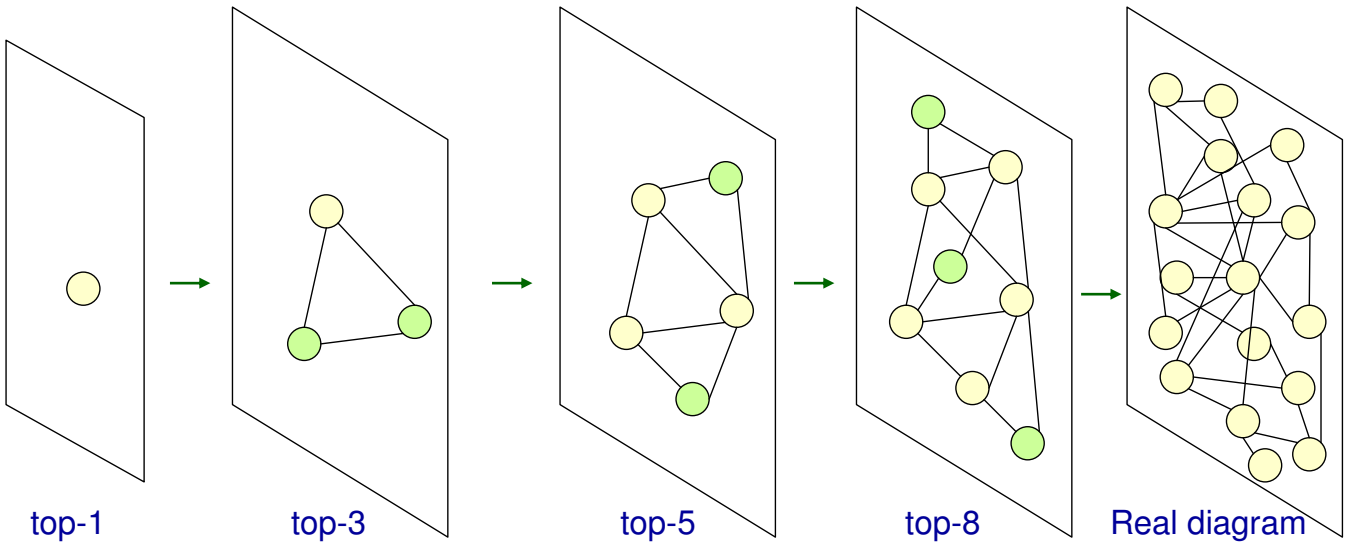


The Idea

Try to rank the elements of the diagram
according to their “importance”

Why ranking can be useful?

- Gradual visualization (understanding): from the most important to the less
  - Provision of top-k diagrams for successive values of k



- Other applications: Keyword searching, ...

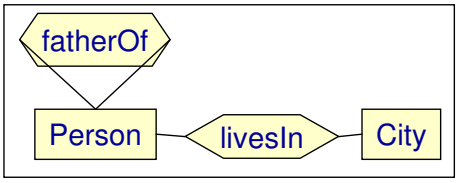
Remark: Web Searching is an Analogous problem

- The Web graph is a very big graph too.
- Link Analysis has been proved very successful in Web searching (and recently in many other domains)
  - Main techniques: PageRank, HITS.

- Idea: Define a PageRank-like scoring scheme for ER diagrams

Applying PageRank on ER diagrams
Differences with the Web

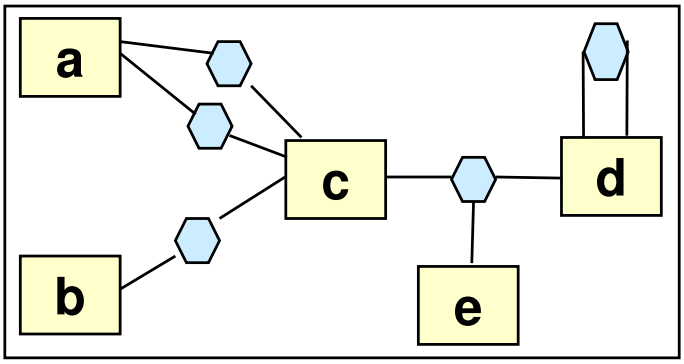
Web	ER
• Directed links	• Undirected relationships
• Binary links	• n-ary relationships
• ignore self hyperlinks	• cyclic relationships are important

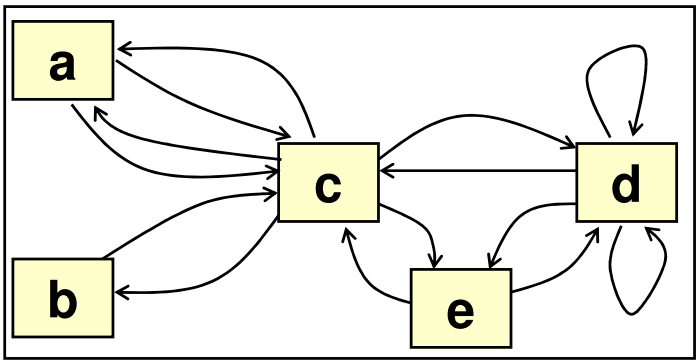


Person should be ranked higher than **City**

Viewing an ER diagram as a Markov chain

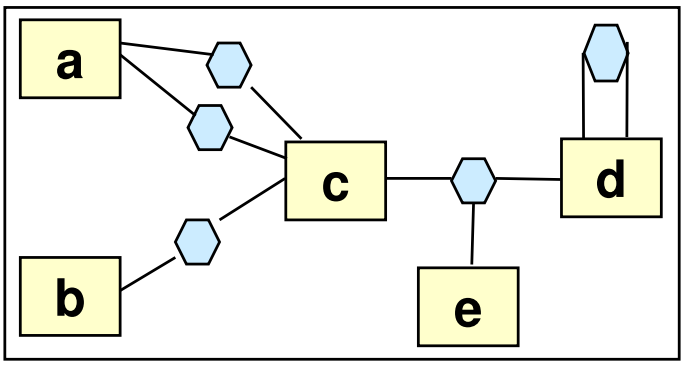
- Entity type => state
- Binary relationship type => two counterpoising transitions
  - An n-ary relationship type is first replaced by n(n-1)/2 binary relationship types





Towards defining EntityRank

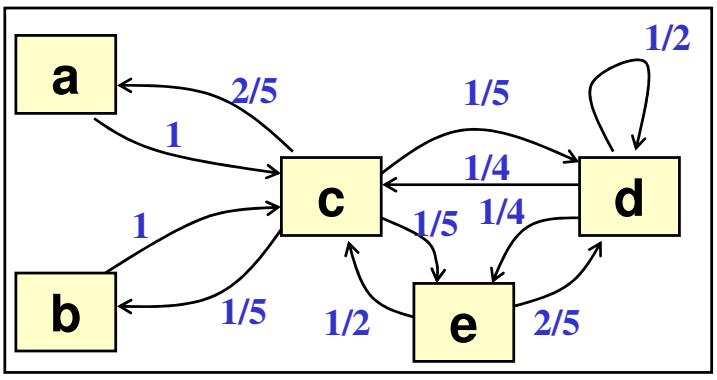
Deriving the Probability Transition Matrix M



A: Adjacency matrix

	a	*b*	*c*	*d*	*e*
a	0	0	2	0	0
b	0	0	1	0	0
c	2	1	0	1	1
d	0	0	1	2	1
e	0	0	1	1	0

By normalizing each row of A to sum to 1



M: transition matrix

	a	*b*	*c*	*d*	*e*
a	0	0	1	0	0
b	0	0	1	0	0
c	2/5	1/5	0	1/5	1/5
d	0	0	1/4	2/4	1/4
e	0	0	1/2	1/2	0

What about ISA Hierarchies ?

- We could view an ISA link as transitions (one way or two-way).
- Our approach
  - We ignore them, but
  - We have an optional preprocessing step where we **collapse** each ISA hierarchy into one node (the root(s) of the hierarchy)
    - that collects all attributes and relationship types

EntityRank

- Suppose a random E-R surfer who at each time step is at some entity type e and then:
  - with probability **q** (e.g. q=0.15) jumps to a **randomly** picked entity type, and
  - with probability **(1-q)** jumps to an entity type that is **connected** with the e
- **EntityRank score := stationary probability**
- This process defines a Markov chain with transition matrix:
  - $q \cdot \mathbf{U} + (1-q) \cdot \mathbf{M}$
  - where $U[e_i,e_j]=1/N$ for all i,j, where N the number of entity types
  - As the transition graph is strongly connected and non-bipartite, the fundamental theorem of Markov chains apply.
  - The score matrix is the principal right eigenvector of the following transition matrix:

    $(q \cdot \mathbf{U} + (1-q) \cdot \mathbf{M})^T$

EntityRank (II)

- The matrix equation gives the following equation for each entity type e:

$$Sc(e) = \frac{q}{N} + (1-q) \sum_{e' \in conn(e)} \frac{Sc(e')}{|\,conn(e')\,|}$$

conn(e): bag (duplicates allowed) with all entities types that are connected with e

B(iased)EntityRank

- The probability of jumping to a random etype is not the same for all, but it depends on the number of its attributes.

- This process defines a Markov chain with transition matrix:
  - q • **B** + (1-q) • **M** where $\mathrm{B}[e_i, e_j] = \frac{|\,attrs(e_j)\,|}{|\,\text{all attributes}\,|}$

$$Sc(e) = q\frac{|\,attrs(e)\,|}{|\,\text{allattrs}\,|} + (1-q) \sum_{e' \in conn(e)} \frac{Sc(e')}{|\,conn(e')\,|}$$

- BEntityRank is a well founded method for incorporating external domain knowledge, and preferences:
  - Number of tuples in the associated database tables
  - Application programs' calls
  - User feedback while interacting with top-k diagrams

Evaluating Ranking Methods for Conceptual Diagrams

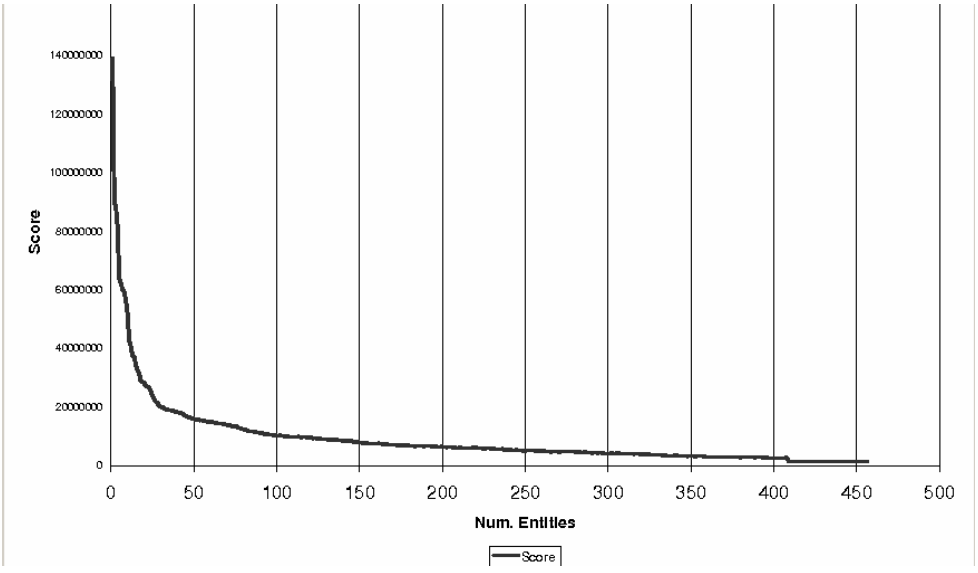
Evaluation is difficult.

The most safe evaluation is to test it on well known schemas

Kinds of Evaluation

- **Empirical**
- **Formal** (TREC/INEX-like)

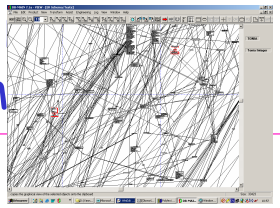
Empirical Evaluation of Ranking Methods for Conceptual Diagrams
Typical Distribution of Scores

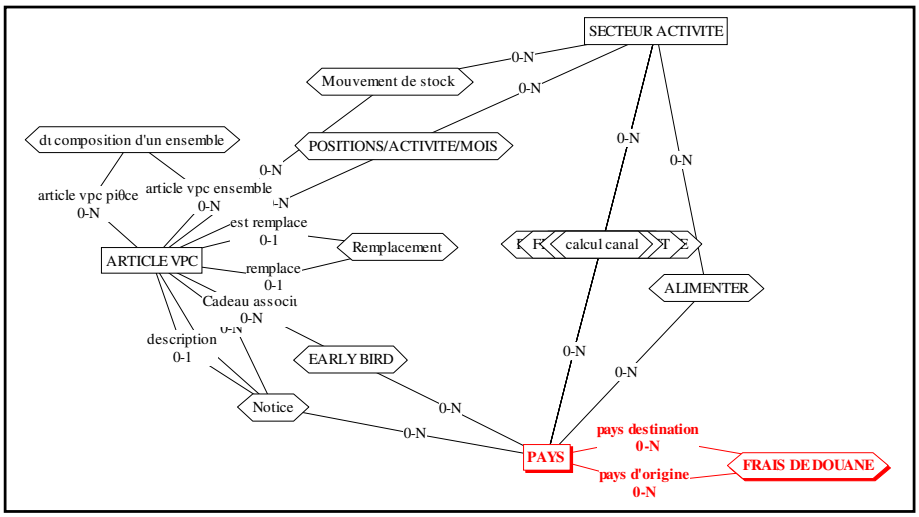


Large ER diagrams tend to have a well-connected kernel

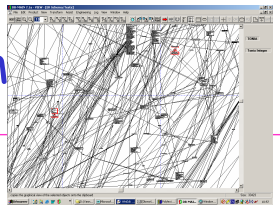
Ziph's Law

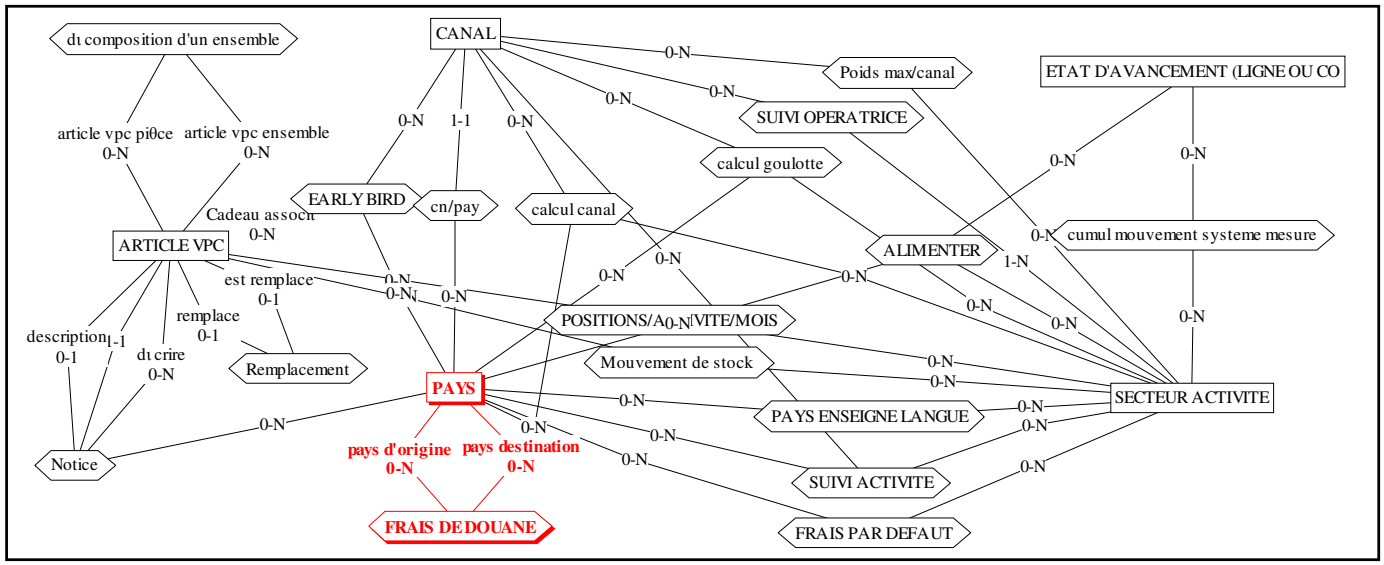
Emp. Evaluation on Top-3 diagram



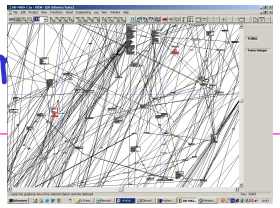


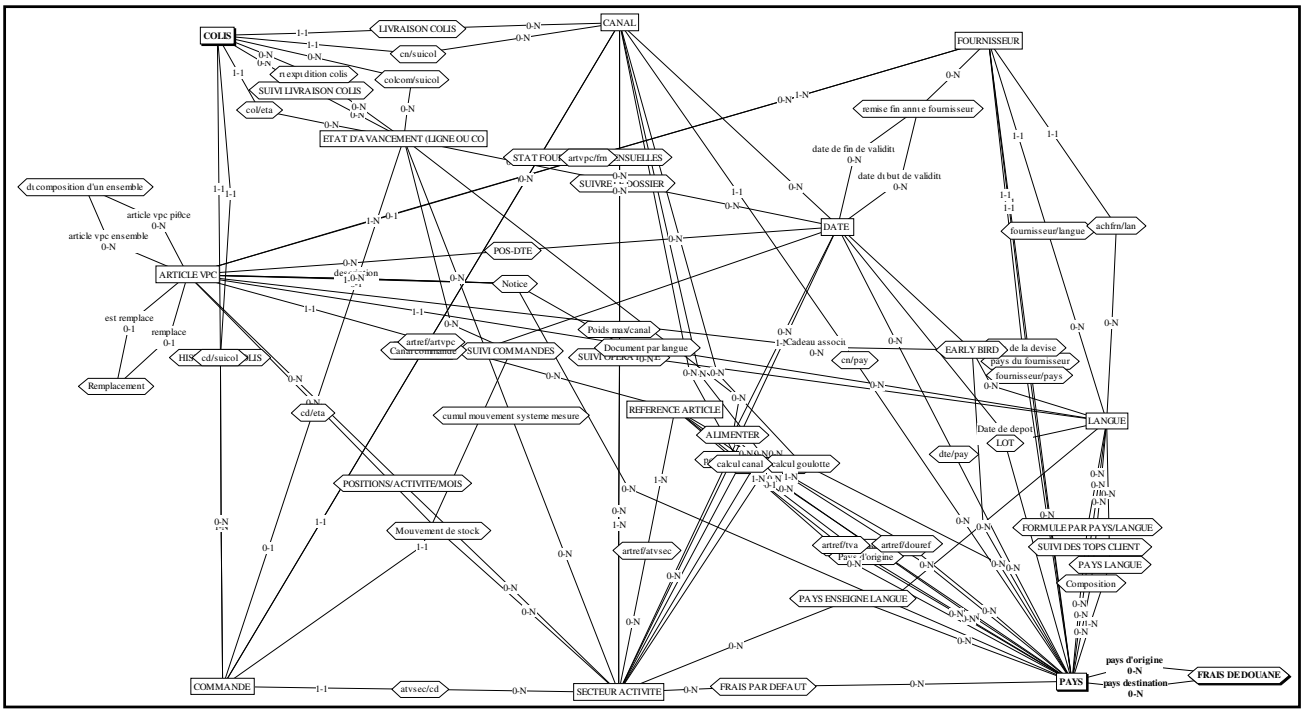
Top-5 diagram





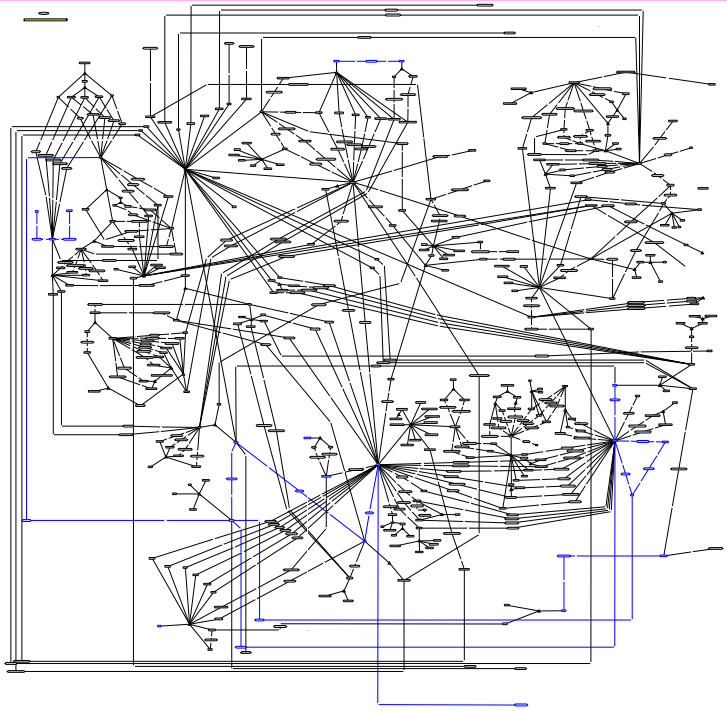
Top-11 diagram



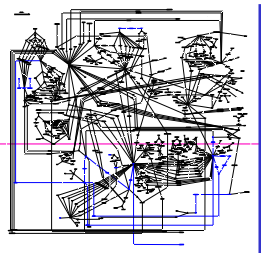


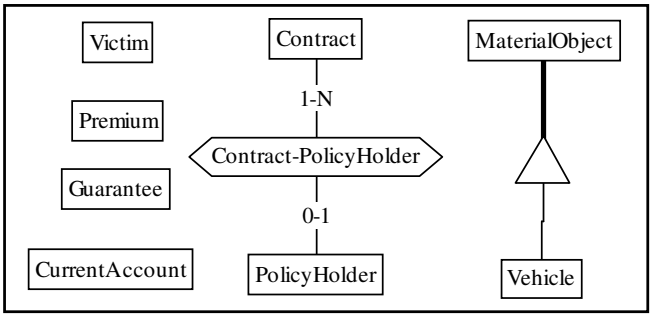
Emp. Evaluation on the TELEBIB schema

Belgian national standard for exchanging messages between insurance companies
(339 entity types and
232 relationship types)

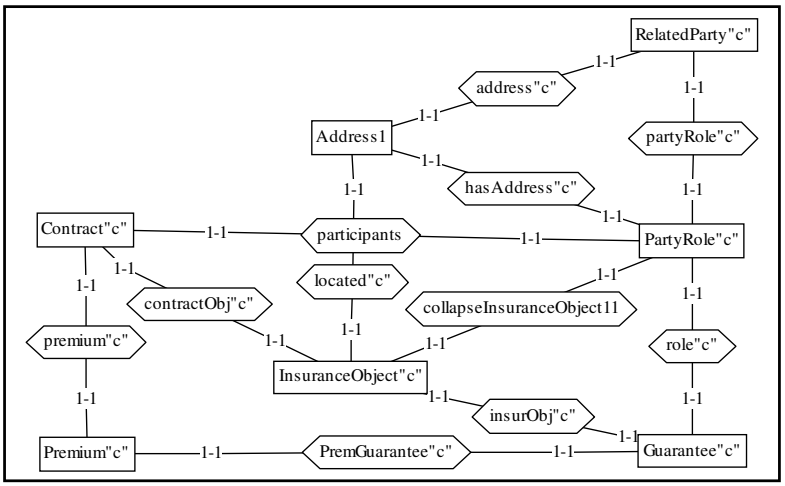


Emp. Evaluation on the TELEBIB schema





Top-8



Top-7 after first collapsing isA hierarchies

Formal Evaluation Methods (TREC/INEX-like)

Steps:

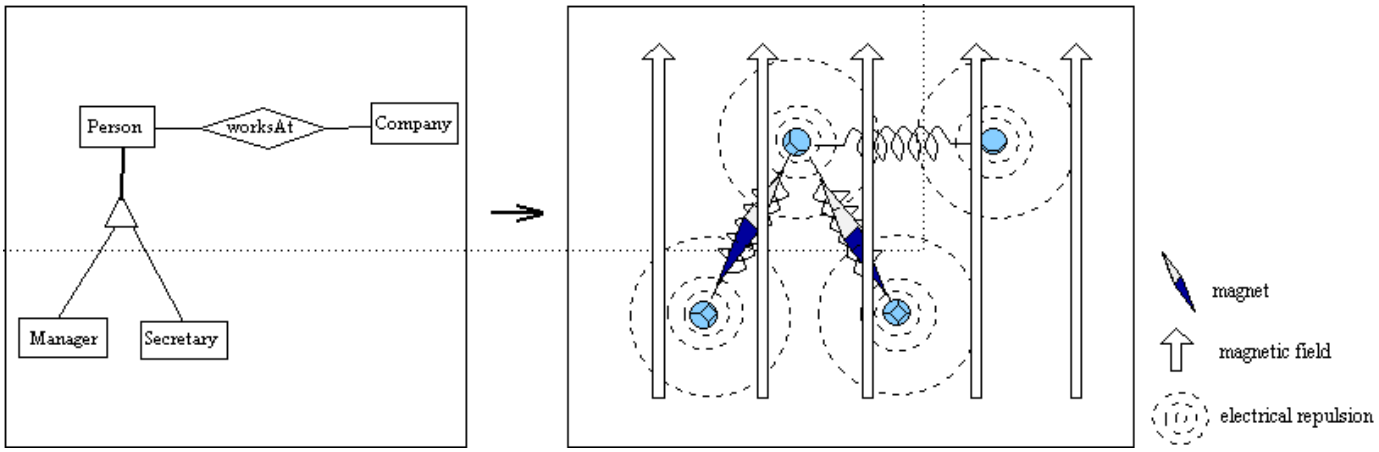
1. Derive the Ideal Ranking of the elements of a schema
   - by aggregating the top-k lists provided by several experts on that schema
2. Compare an automatically derived ranking w.r.t. the ideal ranking based on appropriate metrics
   - ... metrics for evaluating the effectiveness of Information Retrieval Systems

How to visualise these top-k diagrams ?

How to visualise these top-k diagrams ?

- For drawing automatically the top-k diagrams we combine the spring-model [Eades84] with the magnetic-spring model [SugiyamaMisue94] in a way that is appropriate for ER diagrams.
- We view an ER diagram as a mechanical system
  - Force model A
  - Force model B
- Drawing algorithm
  - Algorithm that simulates the mechanical system
  - Its seeks for a configuration with locally minimal energy

Force Model A



Force Model A: Computing the exerted forces

- Force exerted on an entity type:

$$F(e_i) = \sum_{e_j \in CON(e_i)} f(e_j, e_i) + \sum_{e_j \in E, e_i \neq e_j} g(e_j, e_i) + \sum_{e_j \in CONISA(e_i)} h(e_j, e_i)$$

String force from connections

Electrical repulsion from every other particle

Magnetic (**rotational**) force from ISA connections

Force Model A: Configuration parameters

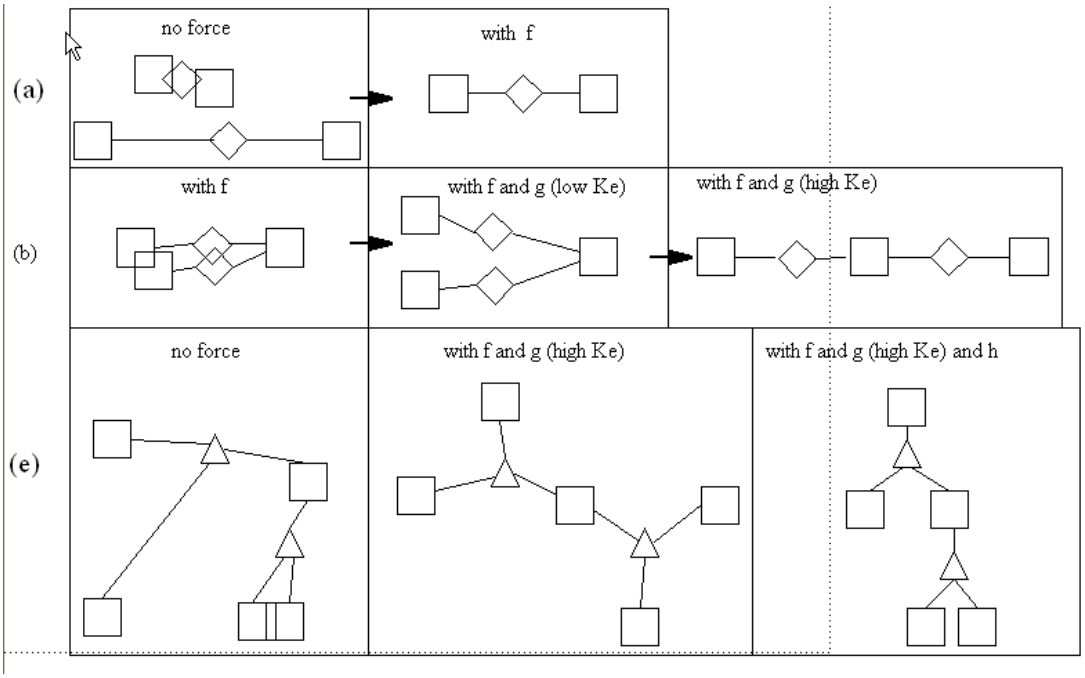
$$F(e_i) = \sum_{e_j \in CON(e_i)} f(e_j, e_i) + \sum_{e_j \in E, e_i \neq e_j} g(e_j, e_i) + \sum_{e_j \in CONISA(e_i)} h(e_j, e_i)$$

$$f_x(e_i) = \sum_{e_j \in CON(e_i)} K^s_{i,j} \left(d(p_i, p_j) - L_{i,j}\right) \frac{x_j - x_i}{d(p_i, p_j)}$$

$$g_x(e_i) = \sum_{e_j \in E, e_j \neq e_i} \frac{K^e_{i,j}}{d(p_i, p_j)^2} \frac{x_i - x_j}{d(p_i, p_j)}$$

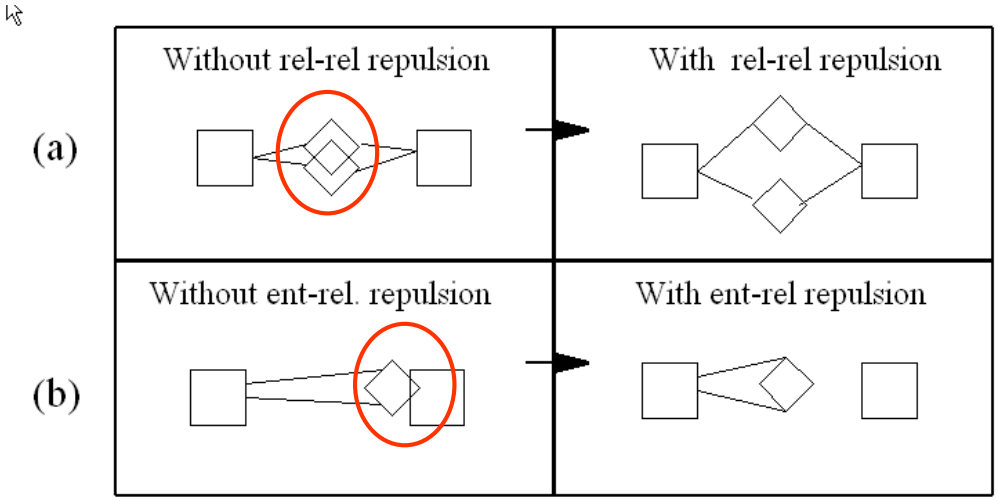
$$h_x(e_i) = \sum_{e_j \in CONNSUP(e_i)} K^m \frac{x_j - x_i}{L_{i,j}} + \sum_{e_j \in CONNSUB(e_i)} K^m \frac{x_j - x_i}{L_{i,j}}$$

The role of the forces *f*, *g*, and *h* and of the configuration parameters

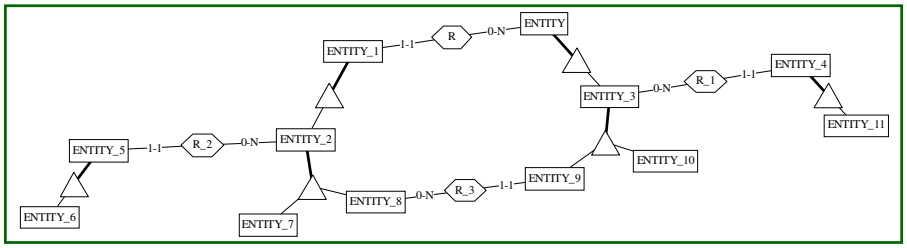


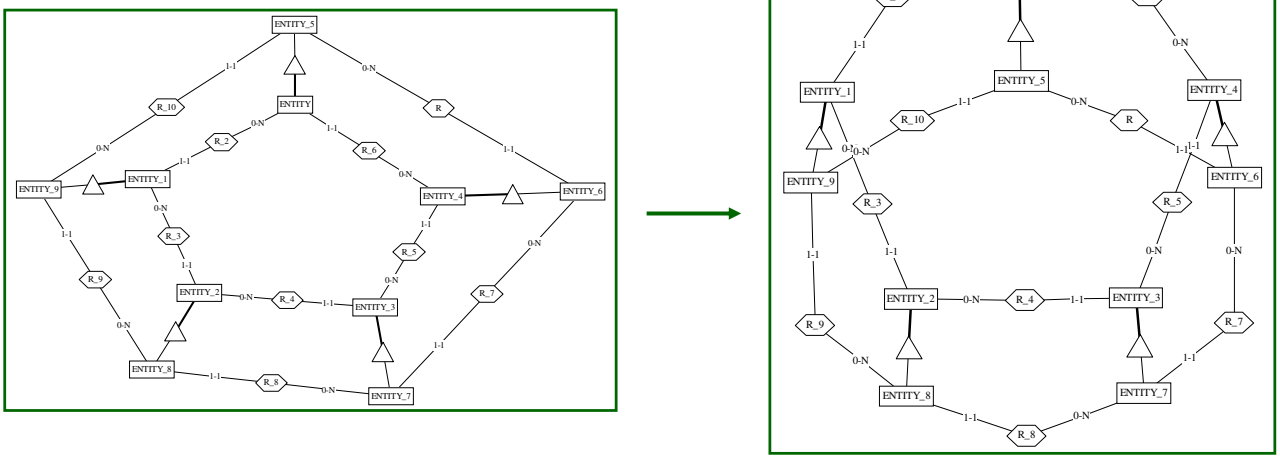
Force Model B

- Relationship types are also considered as particles in order to discourage overlaps.

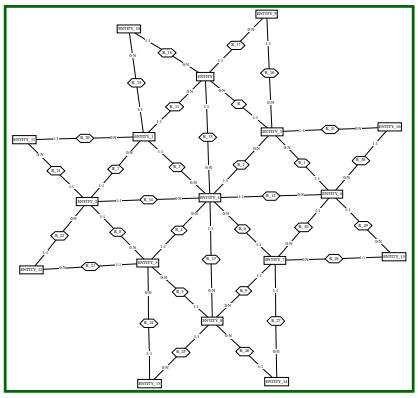


B: Drawing ER diagrams
Experimental Evaluation> Multiple Isa Hierarchies

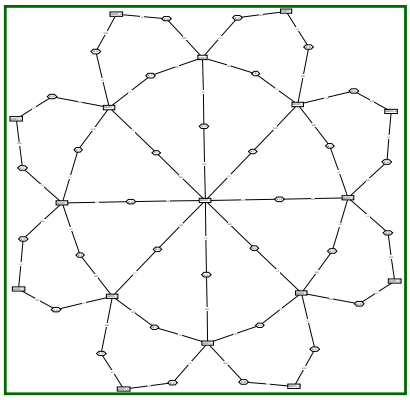




Experimental Evaluation
FM A vs FM B



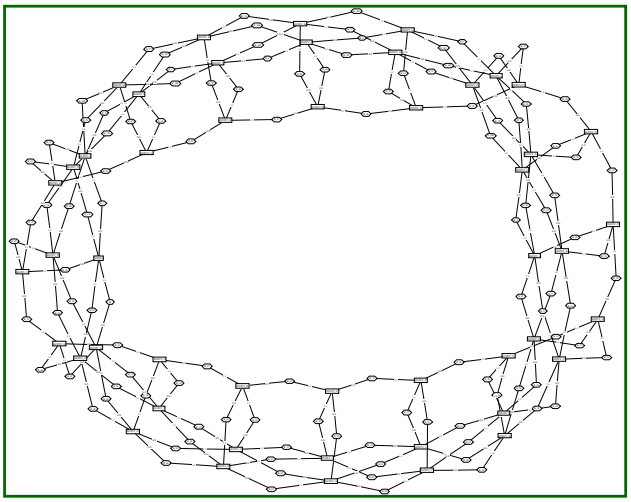
- FM A



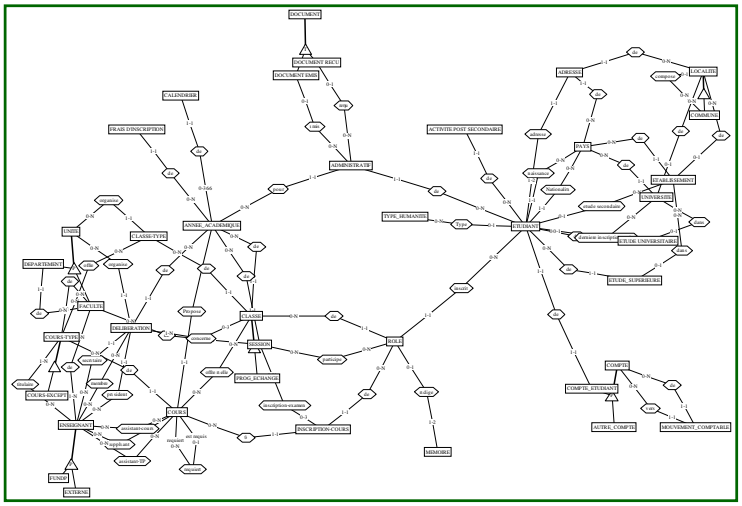
- FM B
  - The tentangles of binary relationship types are unnecessarily not aligned.
  - Computationally more expensive

Experimental Evaluation> On bigger diagrams

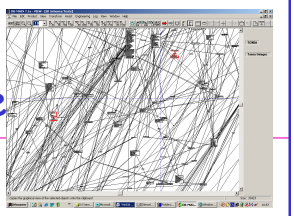
Artificial diagram

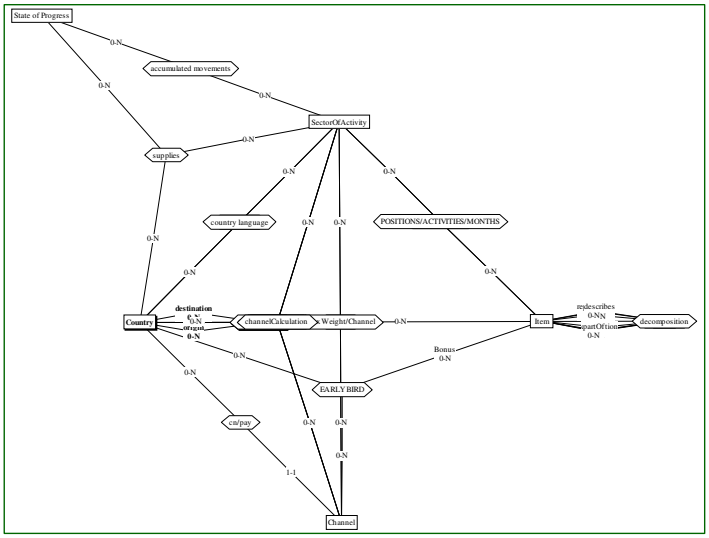


Real ER diagram

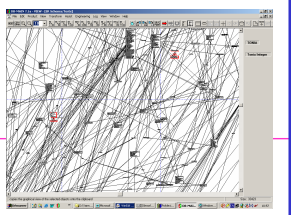


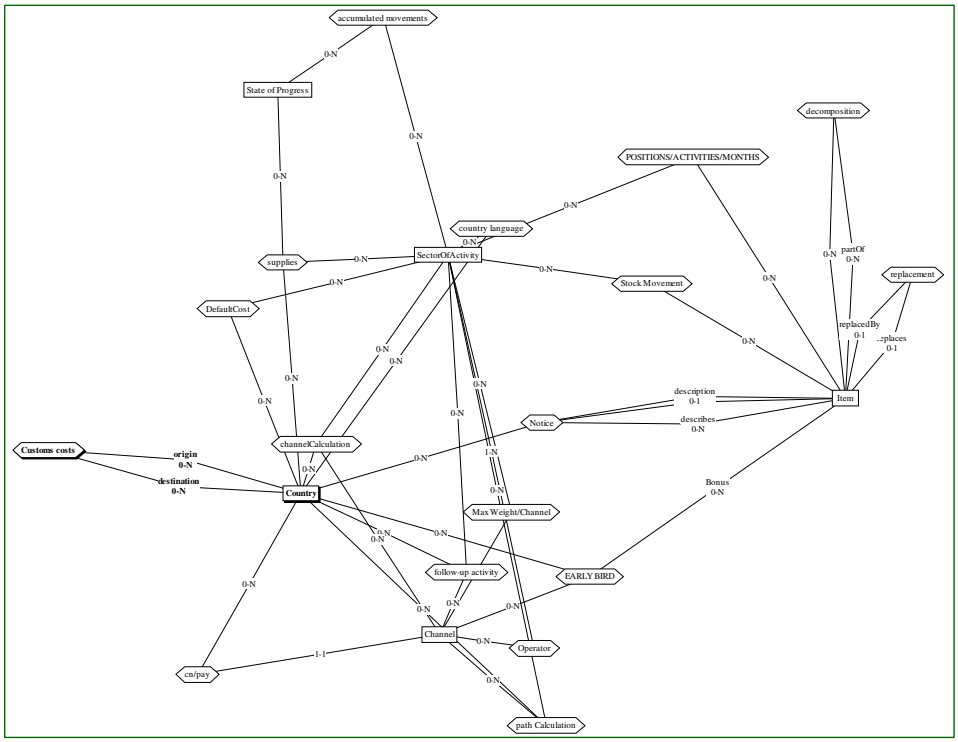
Experimental Evaluation› On real diagrams
Top-5: FM A (after 2 different initial placeme



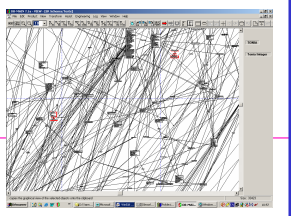


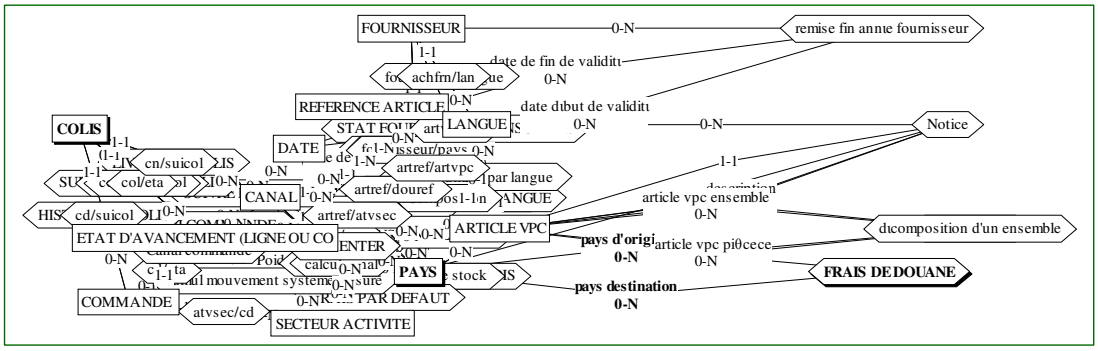
Experimental Evaluation› On real diagrams
Top-5: FM B





Experimental Evaluation› On real diagrams
Top-11: FM B (**dense diagram**)

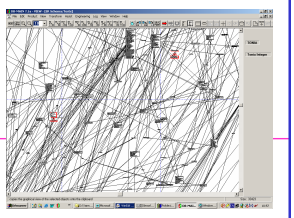




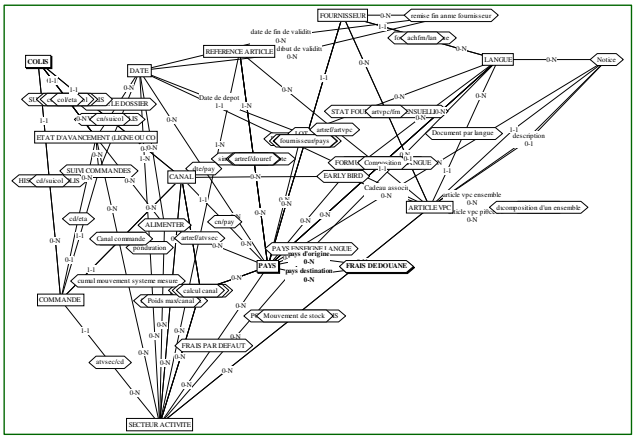
Solutions:

- Play with the configuration parameters
  - local density-based configuration of parameters (future research)
- or just scale up the entire diagram

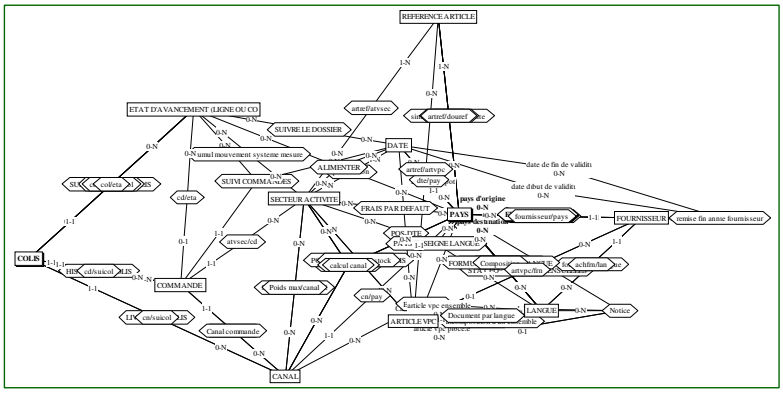
Experimental Evaluation› On real diagrams
Top-11: FM B (**dense diagram**)



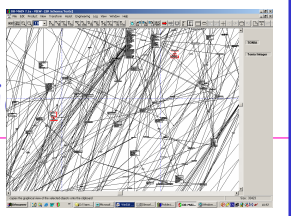
Less stiff springs

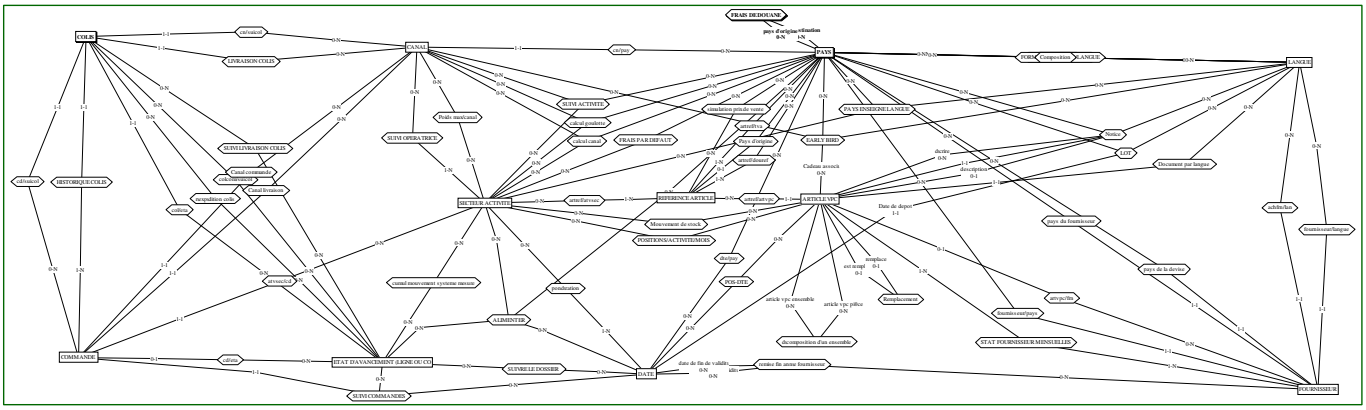


Higher electrical repulsion

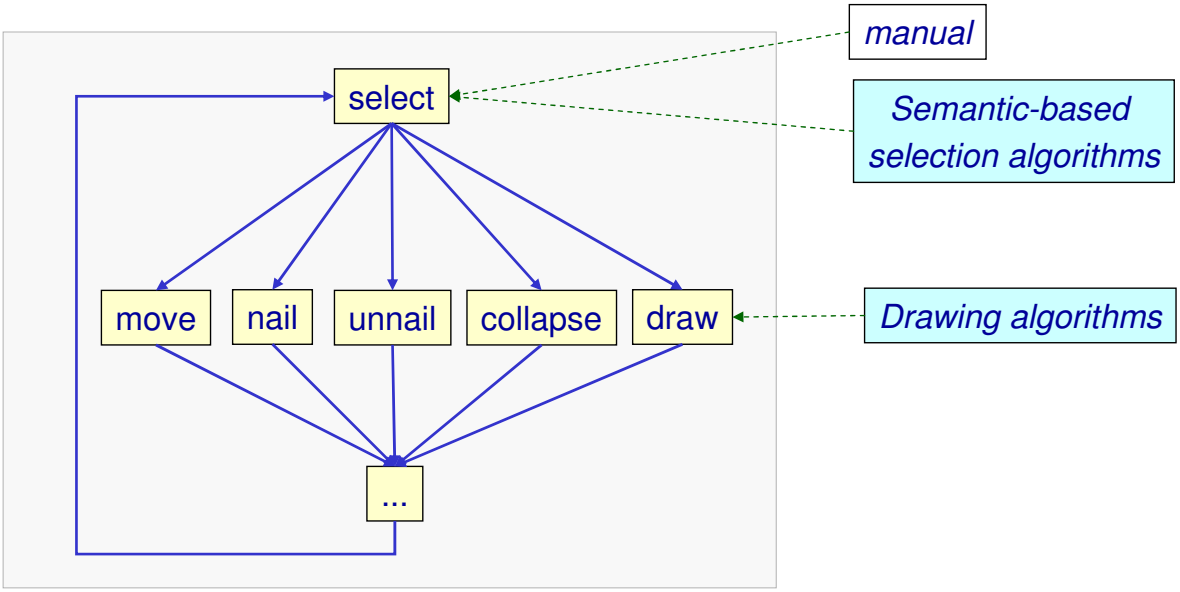


Experimental Evaluation› On real diagrams
Top-11: After manual rectification/beautificati

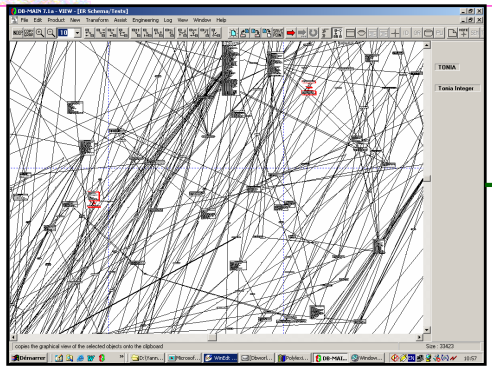




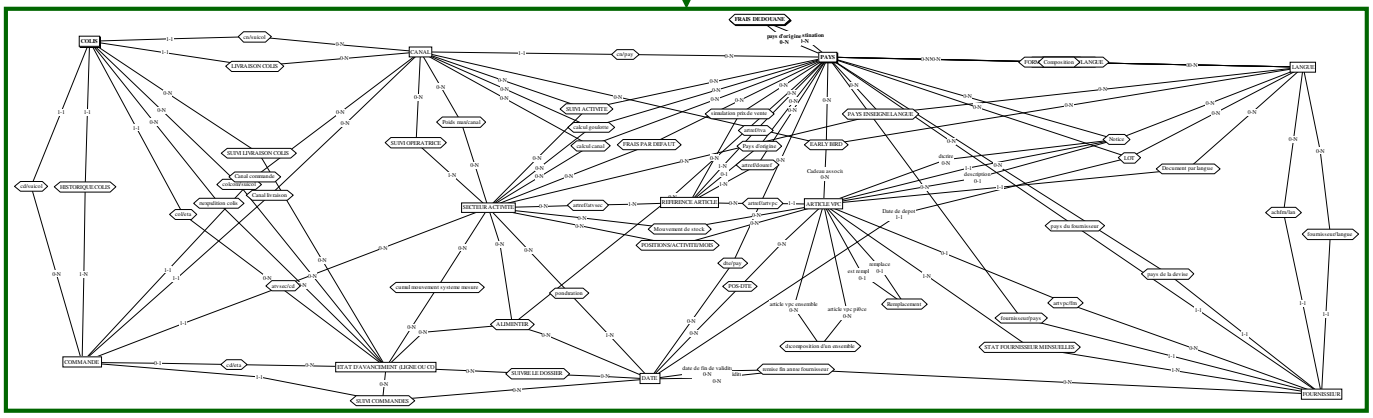
Extending the CASE tool DB-MAIN with a semiautomatic conceptual drawing approach



Concluding Remarks



10 minutes



How to Tame a Very Large ER Diagram
(using Link Analysis and Force-Directed Placement Algorithms)

Concluding Remarks

- The diagrams of large schemas are very difficult to understand and visualize
- *Top-k diagrams* that are derived by **(B)EntityRank** can aid the understanding, the visualization, and the drawing of such diagrams.
- The experiments on real diagrams were successful.
- We can draw top-k diagrams successfully (for small k)

Further research

- Ranking
  - BEntityRank and **User Feedback**
  - **Test Collections** for Evaluations
  - Take into account multiplicity constraints?
- Drawing
  - **Local-Density Adaptation** (EntityRank scores might help)