Information Modeling
UML, ER, NIAM - what is the difference?

Egil P.Andersen
Norwegian Computing Center
P.O.Box 114, Blindern, 0314 Oslo, Norway
Tel: +47 22 85 25 94, Fax: +47 22 69 76 60
Egil.Paulin.Andersen@nr.no

Slides: **http://www.nr.no/~egil/hit-model-101000.zip** (PS: >4Mb)

UML	- (OMG) Unified Modeling Language (*http://www.omg.com/uml*)
ER	- Entity Relationship
NIAM	- Natural language Information Analysis Method (*http://www.orm.net/overview.html*)

Information Modeling Methods

Information Modelling

To model the information in which we are interested for a particular system, i.e., facts, knowledge, etc, about what we perceive to be objects in the system being modelled, and this described as *structural relationships* between the objects involved.

The essence of information modeling is
a) to represent the information of interest to the modeling task
b) to assure consistency with respect to the constraints that apply to the information represented

Information modeling is important! Information persist - applications come and go...

Information Modeling Methods

- ER - the first data or information modeling method (published '76 by Chen).
- NIAM (*) - NIAM is an information modeling method with a rather cumbersome syntax, but its underlying principles are very important and useful to understand the essential characteristics of information modeling (which are syntax independent - and thus ER/UML/NIAM independent!). (NIAM is now mostly known under the name of ORM (Object Role Modeling), which is an extension of the original NIAM to include behavioural object modeling).
- UML - a relatively new object-oriented analysis/design method. Good tool support. OMG (the Object Management Group) is in charge of developing and standardising UML. UML seems to become much of a standard for object-oriented and relational analysis/design.

(*) G.M.Nijssen, T.A.Halpin; *Conceptual Schema and Relational Database Design - A Fact Oriented Approach*; Prentice-Hall, 1989, ISBN 0-7248-0151-0

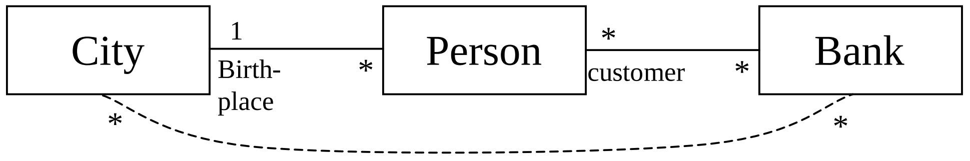
Rule Number 1

A picture may be worth a 1000 words, but
whatever modeling language you use - whatever syntax you prefer - whatever tool you are using always add the 1000 words +/- for each drawing you make!

A model diagram itself is of <u>no value at all</u>
without an elaborate formal or informal description stating how to interpret it, exactly what is expressed by the diagram, and so on.

Example:

In any information model, you can add a many-to-many binary association between *almost any* pair of classes/entities without making any other error than perhaps forgetting to identify those relationships that are derivable.



PS - ikke glem dette på eksamen :-)
(… dersom dere vet hva dere har modellert… hvis ikke - gjør som dere ikke vet dette …)

UML - Unified Modeling Language

Information modeling is just one ingredient in a full system analysis/design process.

UML supports many other analysis/design areas beside information modeling
…but on the downside: UML is huge(!) - not fully consistent - not very precise semantics - areas not supported by tools are mostly of "academic" value - ...

The following are some of the diagram types supported by UML:

Class diagrams: for information modeling and static class/object behaviour modeling

Object diagrams: for exemplifying actual object structures

Use cases: for describing system services as perceived and accessed by its users

Sequence diagrams: dynamic object interaction modeling - message sequences

Collaboration diagrams: dynamic object interaction modeling - object interactions

Statechart diagrams: object state modeling

Activity diagrams: object state modeling

Component diagrams: for implementation - structure of the code - software component dependencies

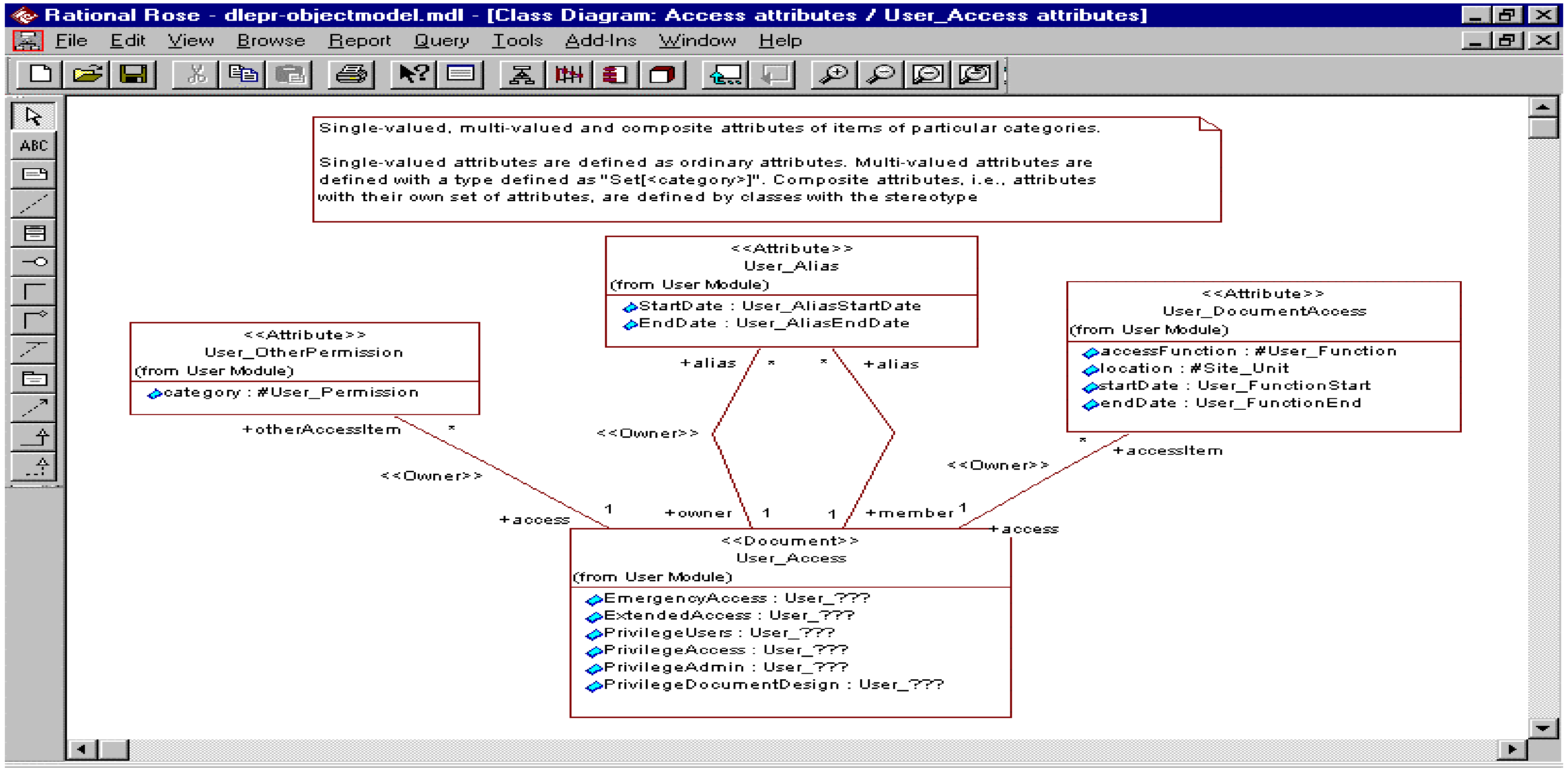
Deployment diagrams: for implementation - structure of the run-time system and its processing

OCL - Object Constraint Language
A predicate based language for defining constraints and business rules.

UML vs ER vs NIAM

UML has some minor syntactical differences from ER (also depending on the ER “dialect”…), but when it comes to information modeling (via UML Class Diagrams), then for all practical purposes there is no principal difference between using UML versus using ER.

The same is “almost” the case for NIAM as well, but NIAM has a construct called “**joint-unique**”, (see later on) which is very useful and quite frequently occuring, that is missing in UML and ER. In general, NIAM is conceptually better at handling n-ary associations where n>2, and thus for doing information analysis.


Rational Rose - dlepr-objectmodel.mdl - [Class Diagram: Access attributes / User_Access attributes]
File Edit View Browse Report Query Tools Add-Ins Window Help
ABC
<<Attribute>>
User_Alias
(from User Module)
StartDate : User_AliasStartDate
EndDate : User_AliasEndDate
<<Attribute>>
User_OtherPermission
(from User Module)
category : #User_Permission
<<Attribute>>
User_DocumentAccess
(from User Module)
accessFunction : #User_Function
location : #Site_Unit
startDate : User_FunctionStart
endDate : User_FunctionEnd
+alias
*
*
+alias
+otherAccessItem
*
<<Owner>>
<<Owner>>
<<Owner>>
*
+accessItem
+access
1
+owner
1
1
+member
1
+access
<<Document>>
User_Access
(from User Module)
EmergencyAccess : User_???
ExtendedAccess : User_???
PrivilegeUsers : User_???
PrivilegeAccess : User_???
PrivilegeAdmin : User_???
PrivilegeDocumentDesign : User_???

Single-valued, multi-valued and composite attributes of items of particular categories.

Single-valued attributes are defined as ordinary attributes. Multi-valued attributes are defined with a type defined as "Set[<category>]". Composite attributes, i.e., attributes with their own set of attributes, are defined by classes with the stereotype

Characteristics of Rational Rose - a UML development tool

+ “Mainstream” - well-known and seen as much of a standard
+ Information modelling and explicit object interaction modelling
+ Object model available via COM/automation - it can be extended and customised!
+ Code generation (but **not** production code…)
+ Informal (can be a plus)

÷ Business rules and behaviour other than explicit object interaction

÷ Conceptual errors cannot be detected - models are not correct/incorrect - no modelling tool can distinguish good from bad models (and this is difficult also for experienced modellers)

÷ Incomplete

÷ Slightly confusing organisation (at least at first…) - quite awkward drawings

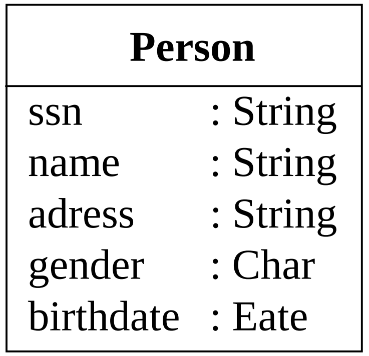
- Consider it mainly as a drawing tool and as a model repository
- Use only those parts that are well understood (or agreed upon within the project), and use it consistently - do **not** “over-model”
- Modeling syntax is not essential, but you are not likely to do e.g. Class Diagrams any better...
- Assuming that analysis/design is essential to large-scale software development, then a modelling tool can be useful to establish good routines for planning and documentation, and as a means for unambigous communication internally and externally.

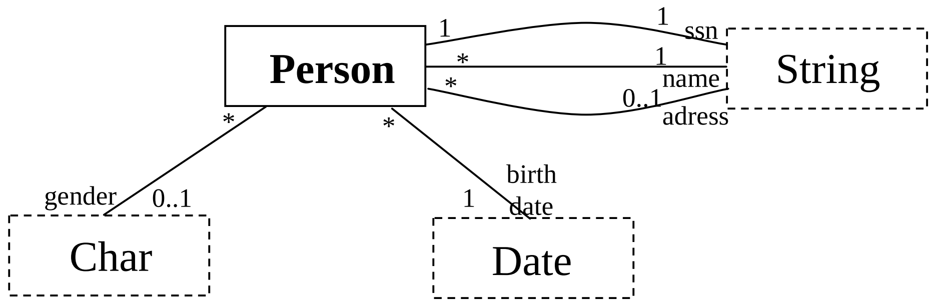
Attributes and Associations

What is an attribute versus an association?

An attribute should be considered as a many-to-one binary association where the opposite class is “suppressed” (i.e., it will not be explicitly implemented).

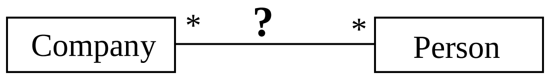
Uniqueness constraints (due to mandatory constraints or “joint unique”) should be specified for attributes similar to how this is specified for associations.





Object-Oriented versus Relational Associations

What if a person can be employed by the same company several times?



C	P
c1	p1
c1	p2
c2	p1
~~c1~~	~~p1~~

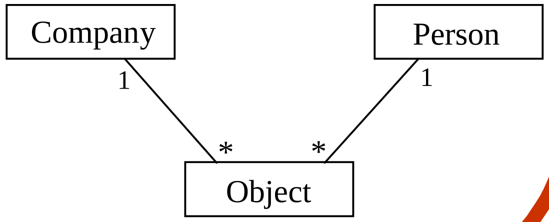
error

object-oriented ”solution”

oid	C	P
1	c1	p1
2	c1	p2
3	c2	p1
4	c1	p1

ok

actual model



Relational vs Object-Oriented Information Models

A mistake

"*We do OO so we do not need those traditional ER-based techniques, normalization and all that, it's irrelevant to us*"

Behaviour modelling - Interaction Modelling

A key characteristic of object-oriented modeling; e.g. by collaboration diagrams or role models.

Relational databases has implicit access routes via joins
Object-Oriented implementations requires explicit object access routes

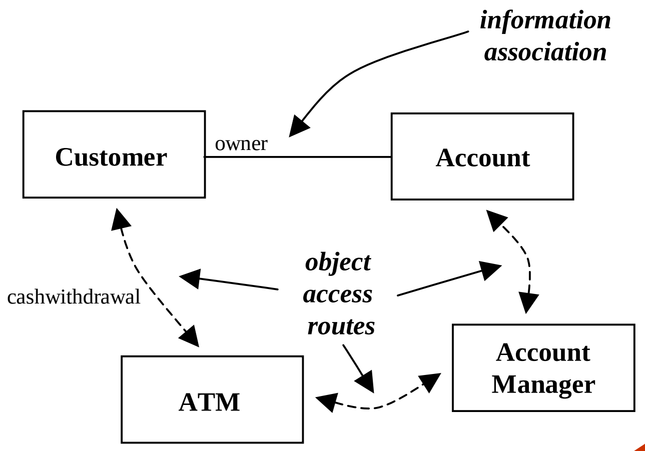
Object Information Associations versus Access Routes

Do **not** mix the two

- How and which information associations to implement is a *modeling decision*
- How and which access routes to implement is an *implementation decision*

The use of information associations is strictly ruled by
the information they represent, and
the constraints that apply to them.

The use of access routes is only concerned with
how to achieve efficient access - they can be added and
removed however it serves the implementation best.



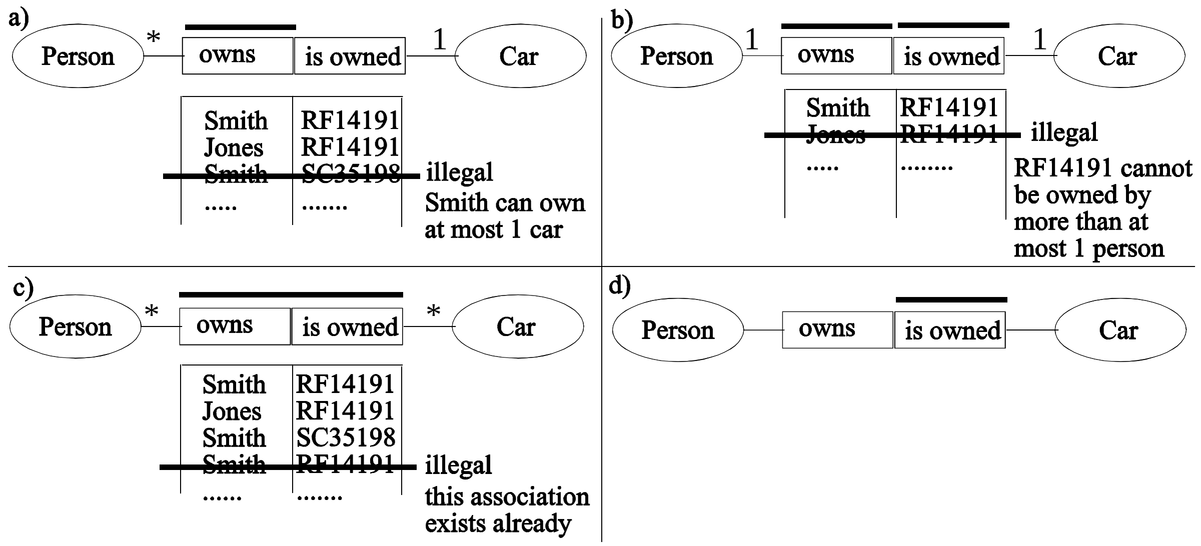
NIAM

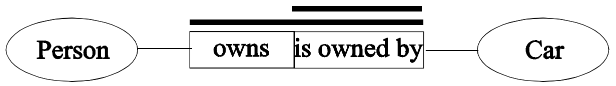
Main characteristics

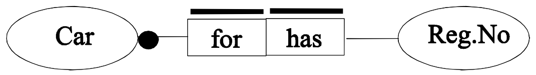
NIAM, i.e., its principles, is particularly valuable for *information analysis*, i.e., to *understand* the information that we are supposed to represent correctly and efficiently within a computer system.

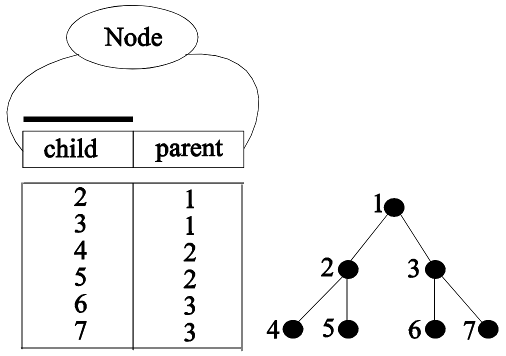
NIAM achieves this by being *“relation-oriented”* rather than *“object-oriented”*
This is a great benefit since it allows us to focus on more manageable subsets of the overall information modeling task - “separation of concern”
How can we possibly know what are appropriate objects for an information domain that we may not really know very well?
Hence, in the beginning, look for **object relationships - not objects**!

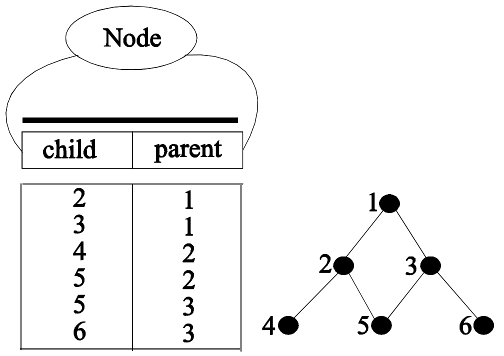
NIAM - Uniqueness Constraints





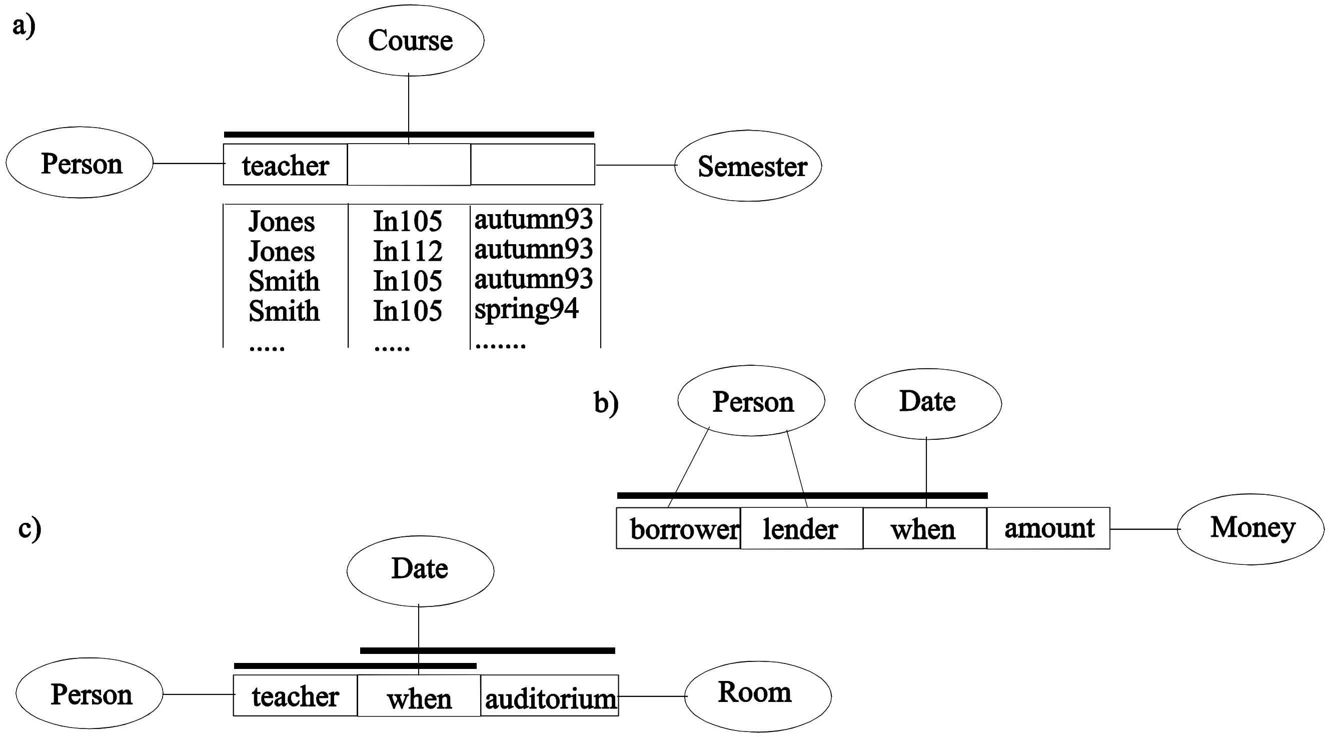






NIAM - Elementary Associations

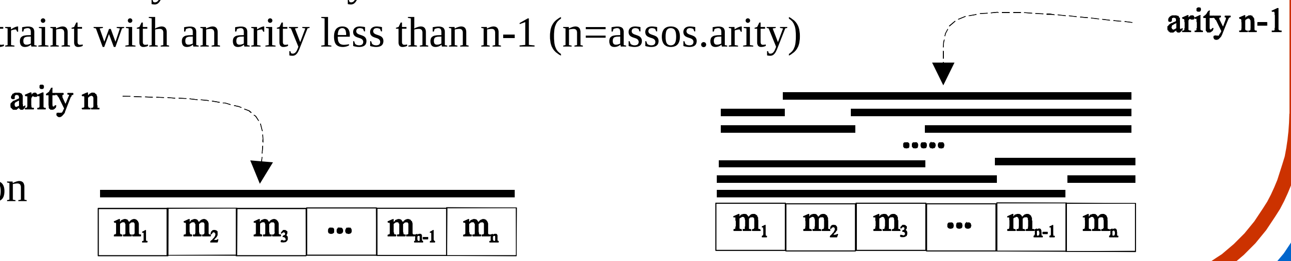
Elementary Association - sufficiently "small" to avoid *"repetition of information"*, but not so "small" that it implies *"loss of information"*.



n-1 rule - an association is elementary if and only if:

1) it has no uniqueness constraint with an arity less than n-1 (n=assos.arity)

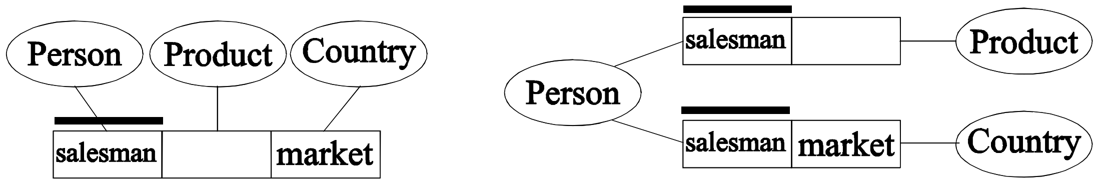
2) there are no other constraints between the members of this association



arity n
m_1 m_2 m_3 ... m_{n-1} m_n
arity n-1
......
m_1 m_2 m_3 ... m_{n-1} m_n

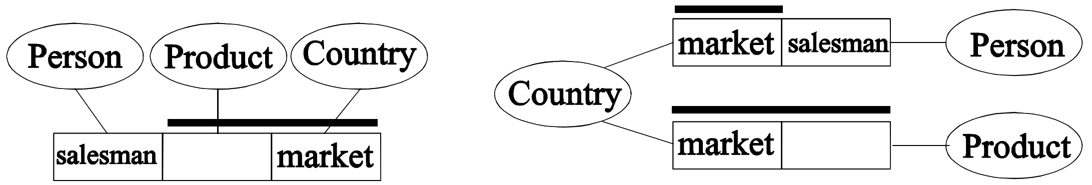
NIAM - Elementary Associations (cont.)

a)



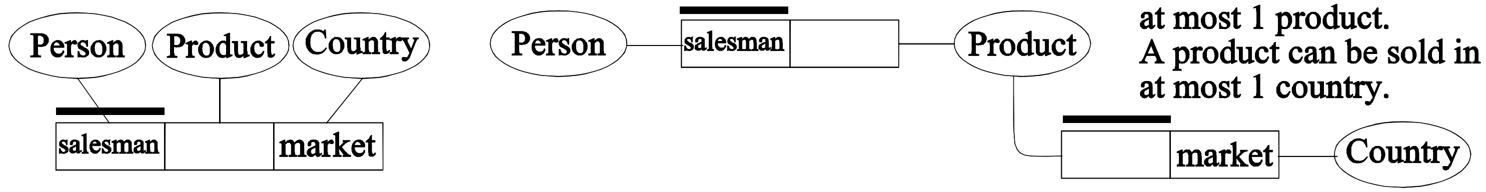
A salesman can sell at most 1 product.
A salesman can sell in at most 1 country,.

b)



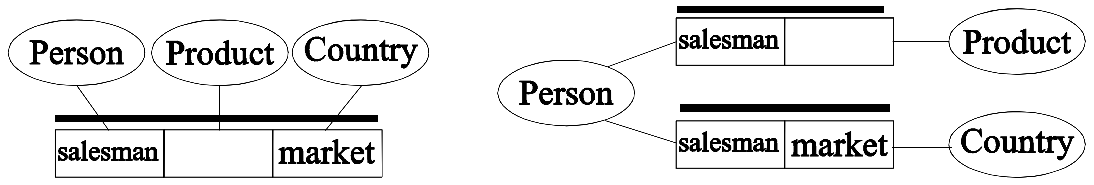
Every country has at most 1 salesman.

c)



A salesman can sell at most 1 product.
A product can be sold in at most 1 country.

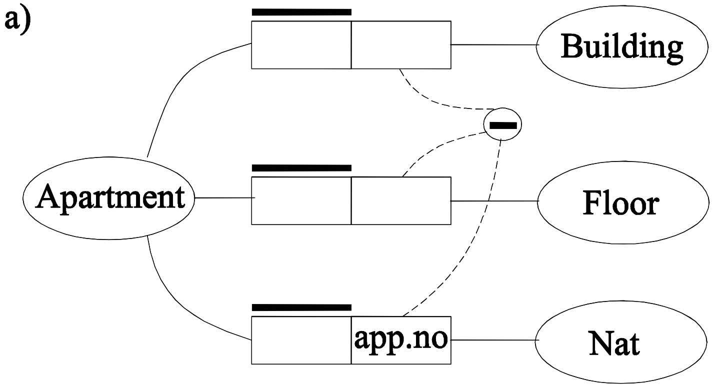
d)



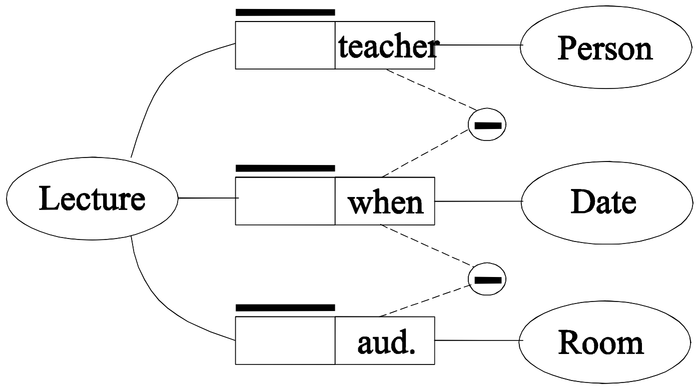
A salesman selling a product will sell it in every country where he sells something.
A salesman selling something in a country will there sell every product he sells.

NIAM - Joint Unique

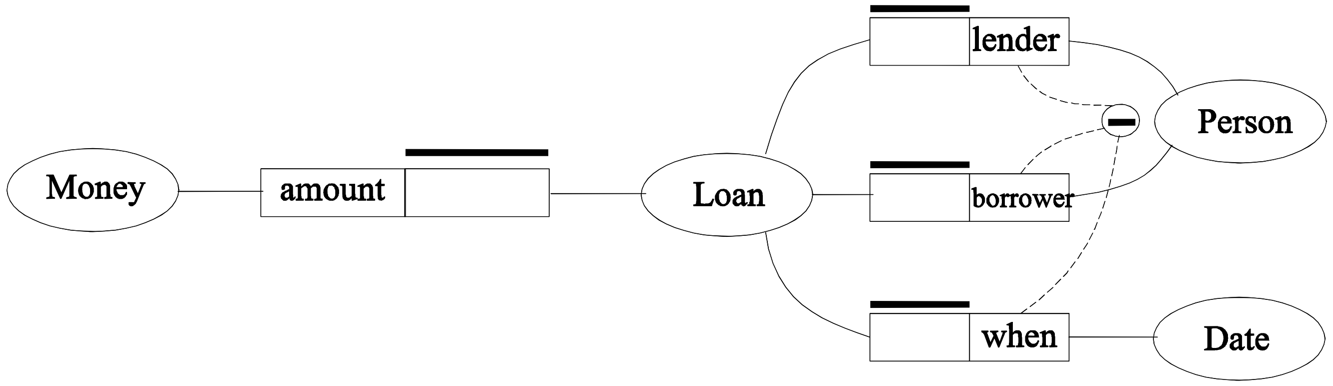
a)



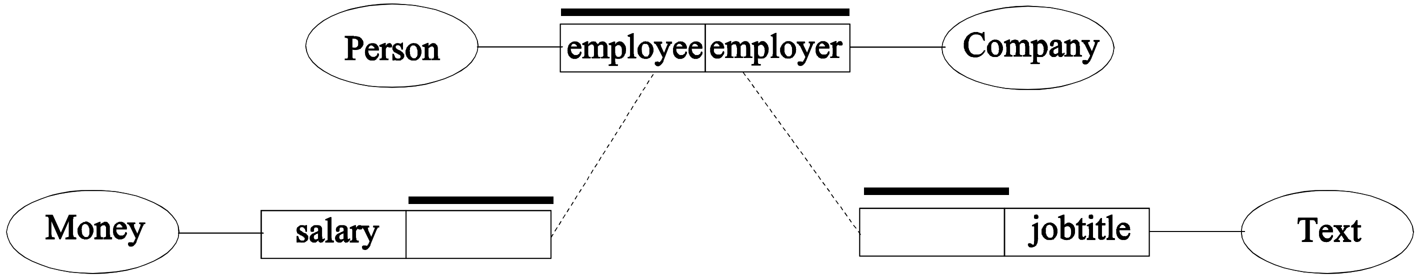
b)



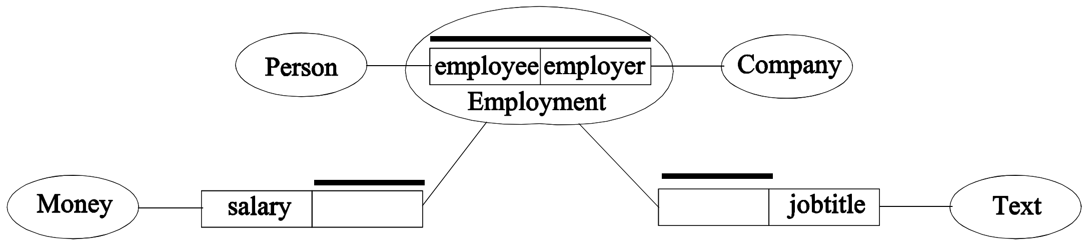
c)



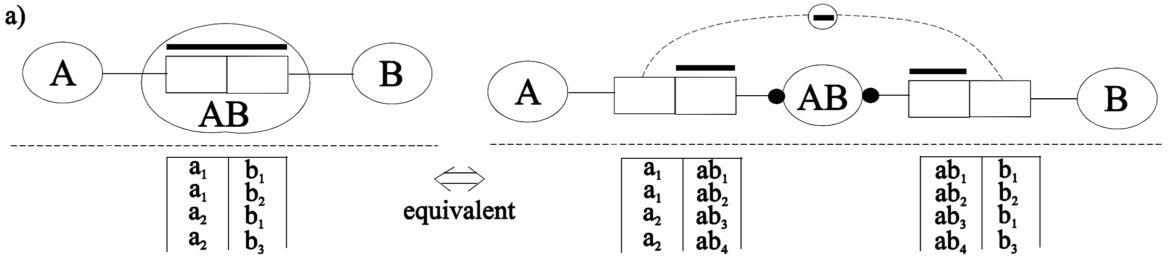
NIAM - Association Associations



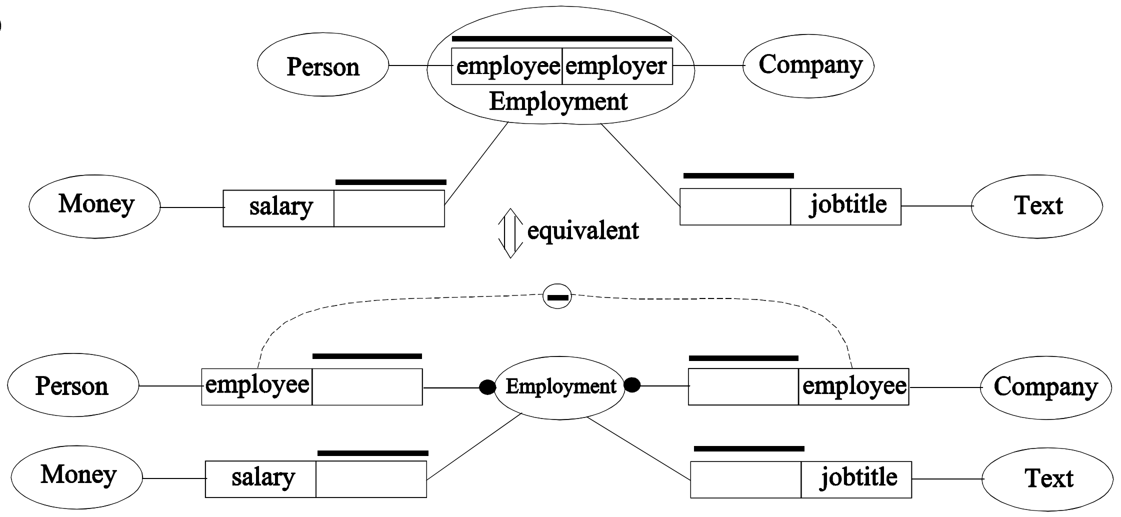
NIAM - Creating Objects from Associations



a)

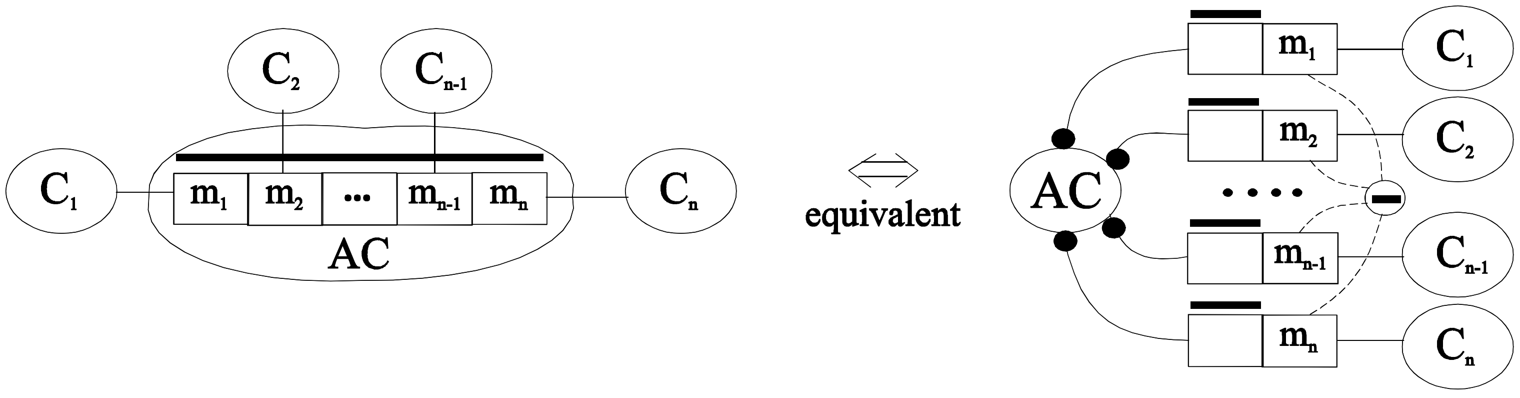


b)

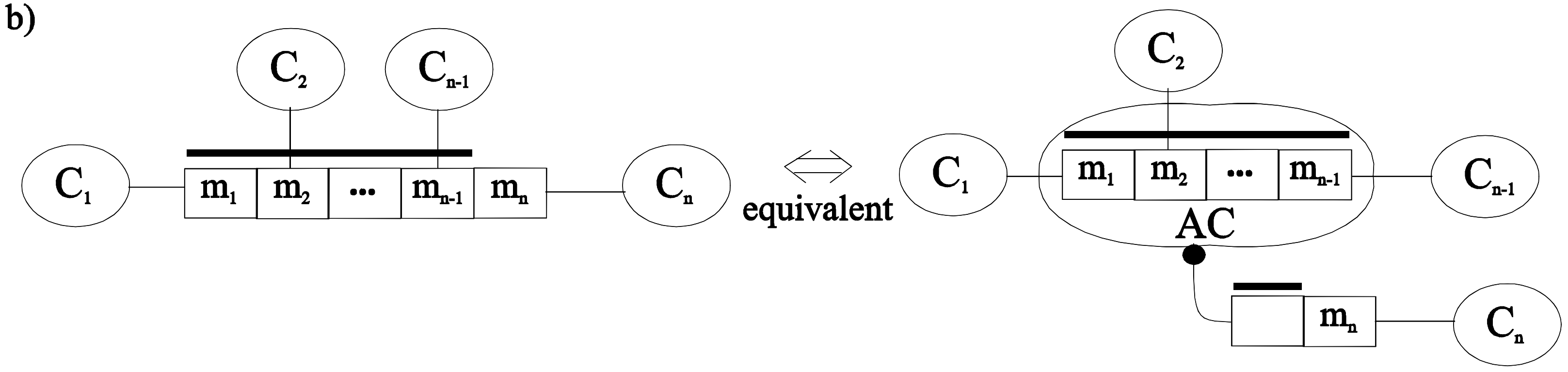


NIAM - from n’ary to Binary Associations

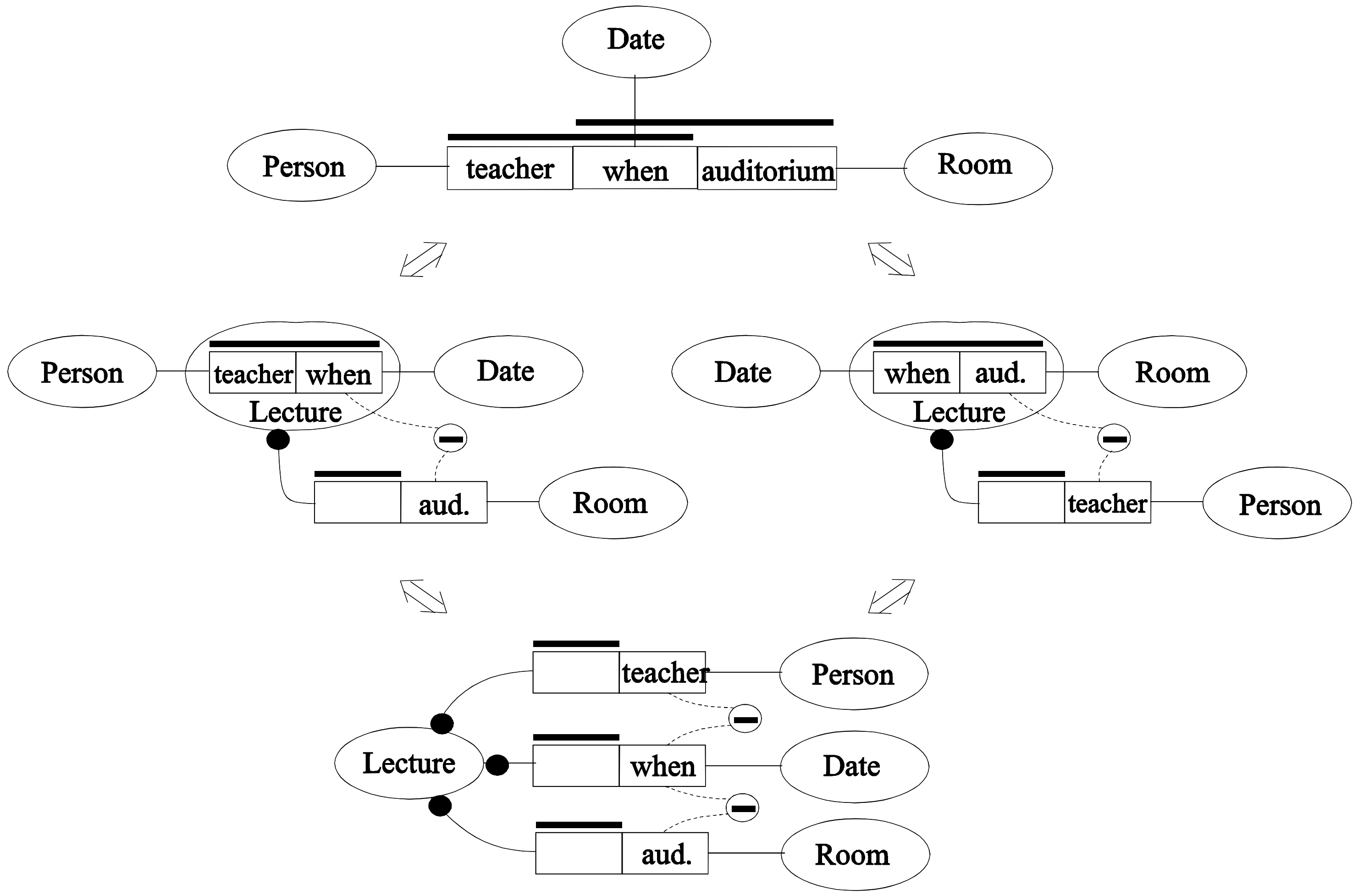
a)



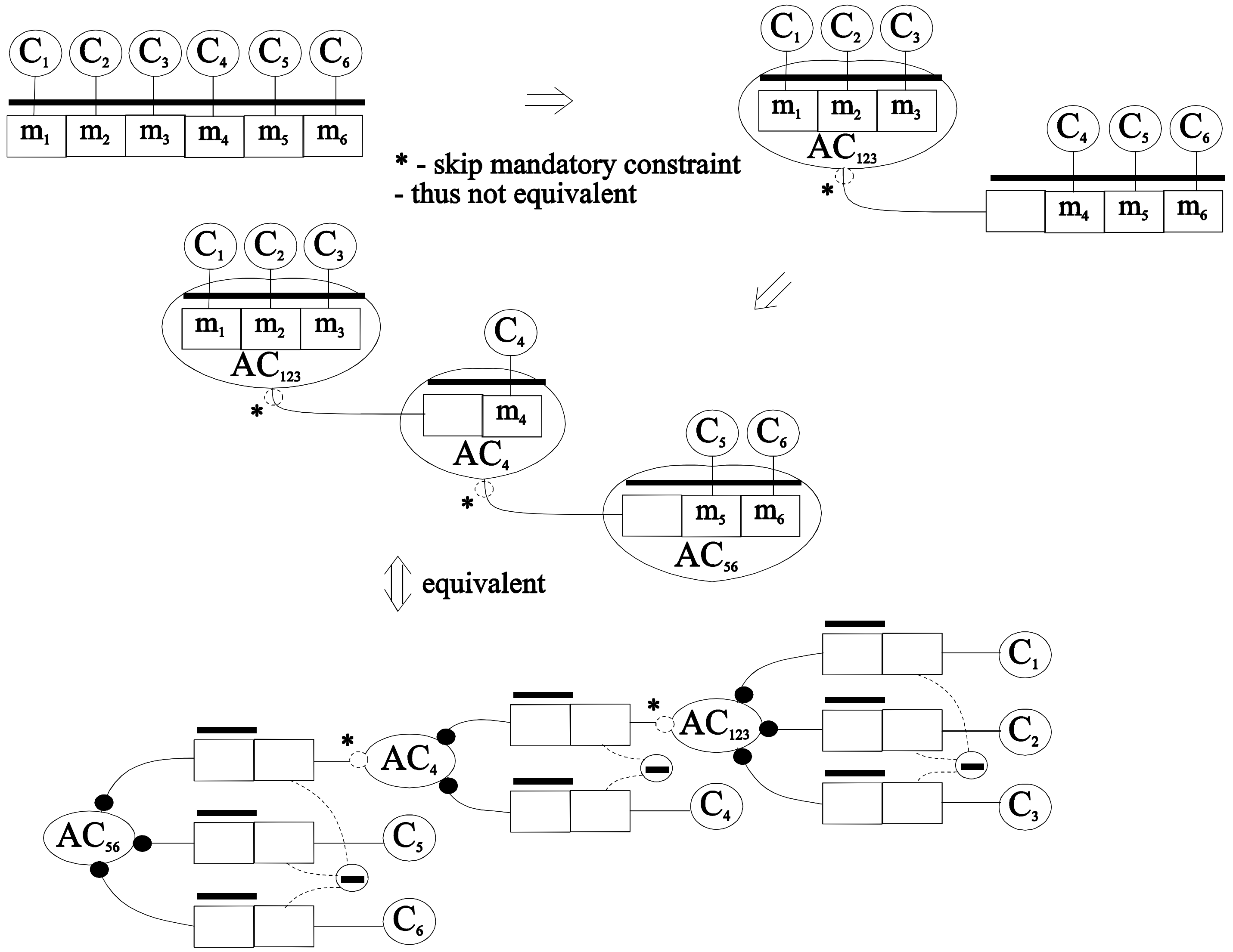
b)



NIAM - from n’ary to Binary Associations (cont.)



NIAM - from n’ary to Binary Associations (cont.)

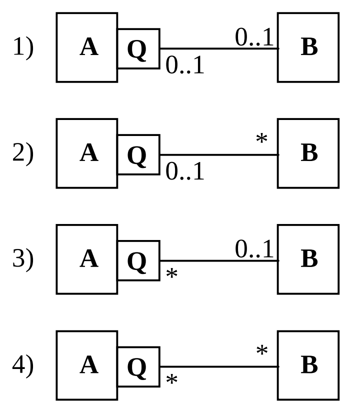


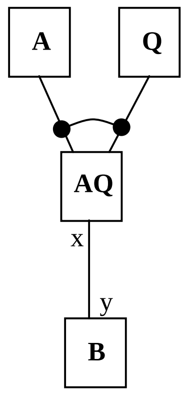
Some UML constructs in NIAM

Qualified Associations are binary associations where a *qualifier* is added to one of the objects involved in the association. Assume having a qualified association between class A and B where a qualifier Q is added to A. The qualifier Q is used to partition the set of B objects associated to a particular A object into disjoint subsets.

Qualified associations are often used to model dictionary-like constructs with the qualifier as index.

NIAMs **joint-unique** is more generally useful than qualified associations (as a special case).



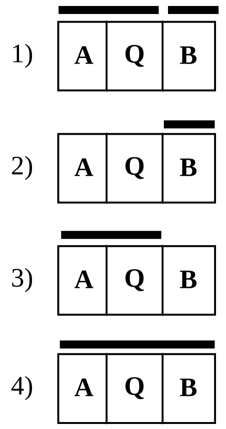


1) x = 0..1
   y = 0..1

2) x = 0..1
   y = *

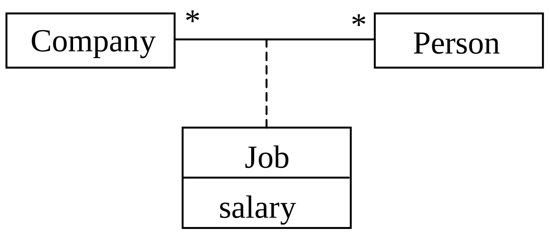
3) x = *
   y = 0..1

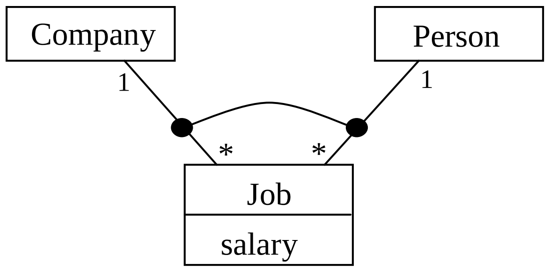
4) x = *
   y = *



Association Classes

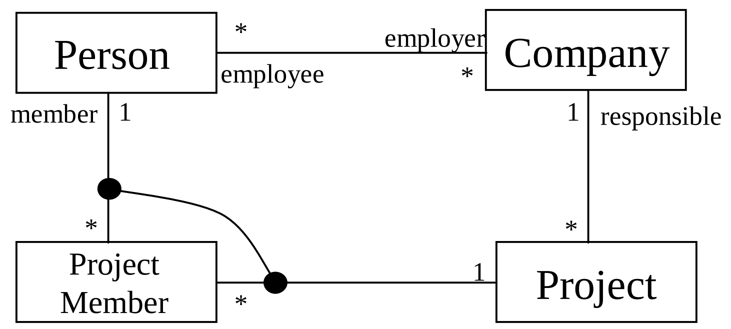
UML association classes are classes representing properties of associations themselves.

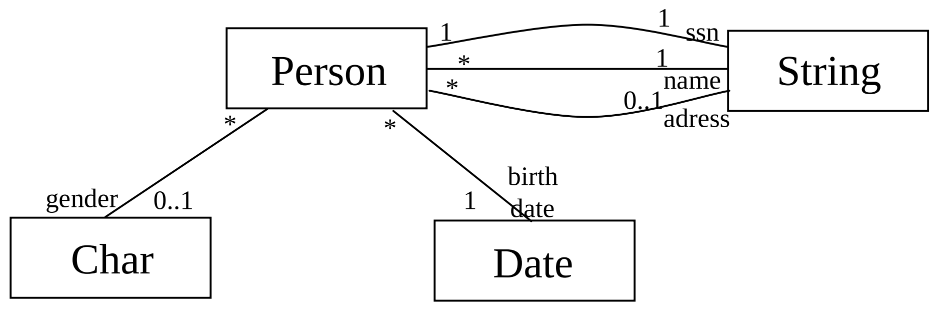




A convention for presenting Information Models

There are usually different subsets of the overall set of associations that are more strongly related than others, or that should be considered together to better grasp the overall structure that they represent.
Sets of associations that are more closely related should be presented separated from other associations that they are not intimately related to.
Obviously, this is a matter of judgment on a case by case basis.





The separate presentation of different subsets of an overall information model is orthogonal to the classes involved.
Thus the same class can be illustrated several times within different associations, and this can make it difficult toget an overview of every association in which a particular class is involved.
However, a tool supporting information modeling can automatically produce a model view where every association of a particular class is illustrated, but, due to no knowledge of the semantics involved, it cannot automatically produce a model split as by the above convention.

Different models for different purposes

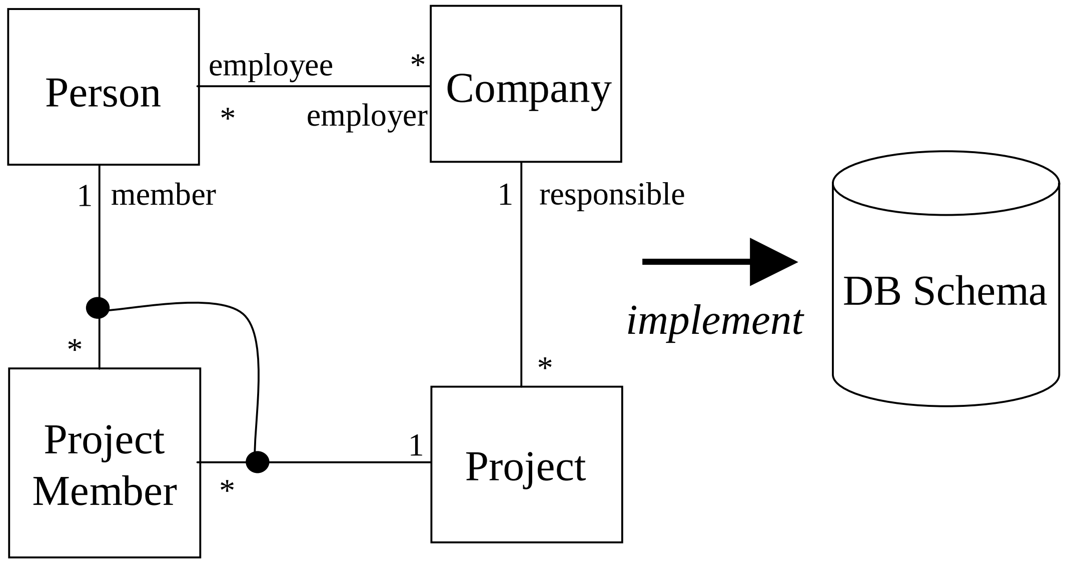
Motivation

To characterize two kinds of information models (IM's) that play different roles in the design, implementation and documentation of a set of related software components.

This to avoid that implementation-oriented issues (at least to a lesser degree) "clutters up" the conceptual view provided to clients working with these components.

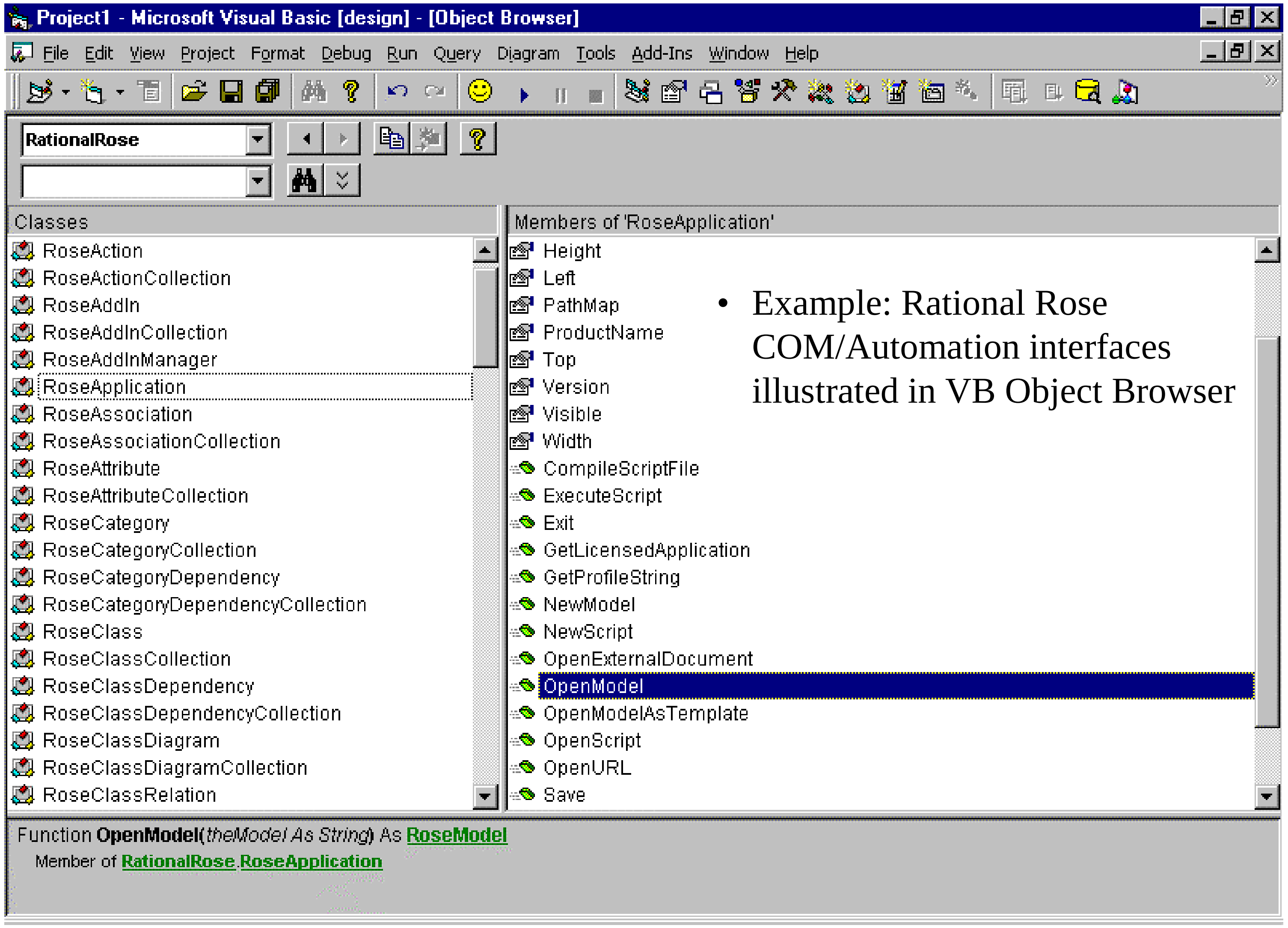
"Traditional" Information Modeling

- In database terminology an IM is a schema; e.g. consisting of tables, columns, keys, etc, in a relational database.
- In logical database design an IM is expressed as an ER-like model, consisting of entities, attributes of entities and relationships between entities.



Component Object Models

- In component systems an object model consists of classes, interfaces, functions, etc, typically specified by an IDL.



Project1 - Microsoft Visual Basic [design] - [Object Browser]

File Edit View Project Format Debug Run Query Diagram Tools Add-Ins Window Help

RationalRose

Classes	Members of 'RoseApplication'
RoseAction	Height
RoseActionCollection	Left
RoseAddIn	PathMap
RoseAddInCollection	ProductName
RoseAddInManager	Top
RoseApplication	Version
RoseAssociation	Visible
RoseAssociationCollection	Width
RoseAttribute	CompileScriptFile
RoseAttributeCollection	ExecuteScript
RoseCategory	Exit
RoseCategoryCollection	GetLicensedApplication
RoseCategoryDependency	GetProfileString
RoseCategoryDependencyCollection	NewModel
RoseClass	NewScript
RoseClassCollection	OpenExternalDocument
RoseClassDependency	OpenModel
RoseClassDependencyCollection	OpenModelAsTemplate
RoseClassDiagram	OpenScript
RoseClassDiagramCollection	OpenURL
RoseClassRelation	Save

Function **OpenModel**(*theModel As String*) As RoseModel
Member of RationalRose.RoseApplication

- Example: Rational Rose COM/Automation interfaces illustrated in VB Object Browser

Implementation versus Interface Information Models

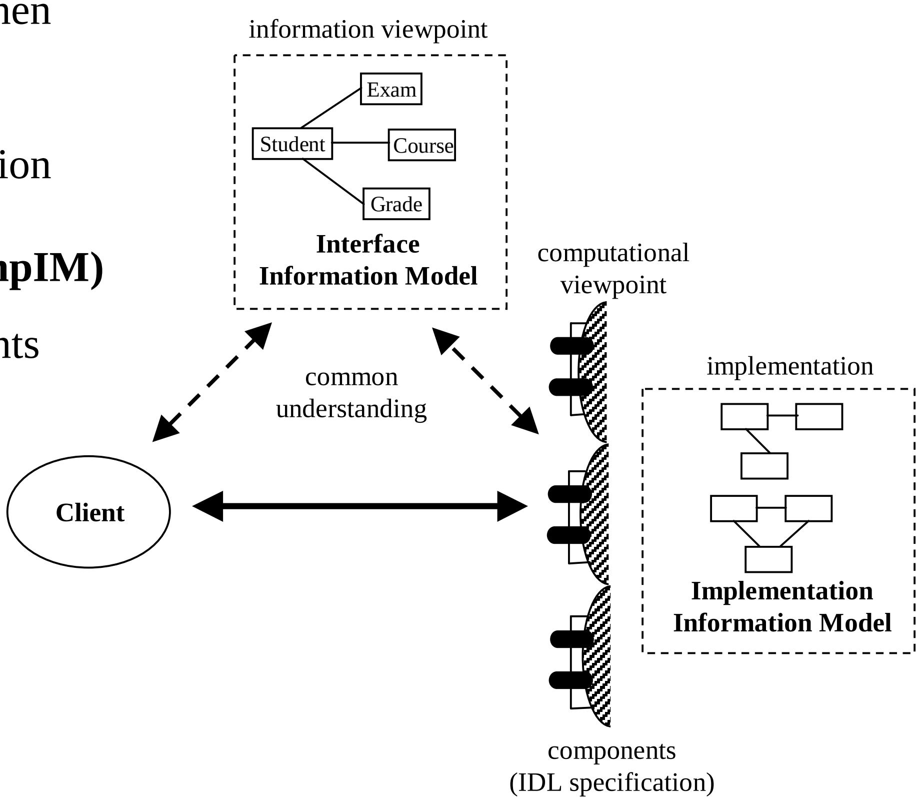
Two different kinds of IM for component-oriented systems where component implementations are encapsulated behind interfaces of functions offered to their clients

Interface Information Models (IntIM)

Conveys the common understanding necessary between a client and a set of related components by describing which objects are made available by the components,
which information must be provided when invoking a function,
which information will be received,
what is the effect of invoking this function

Implementation Information Models (ImpIM)

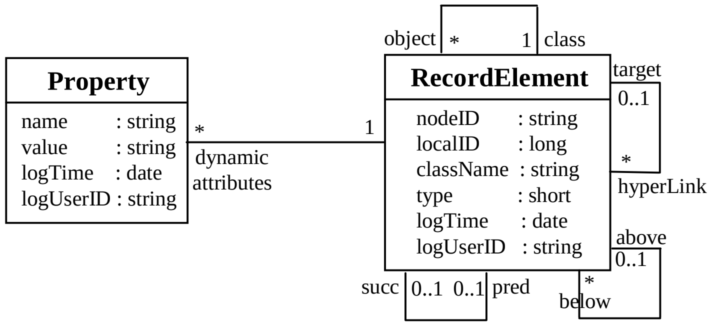
A basis for implementing the components



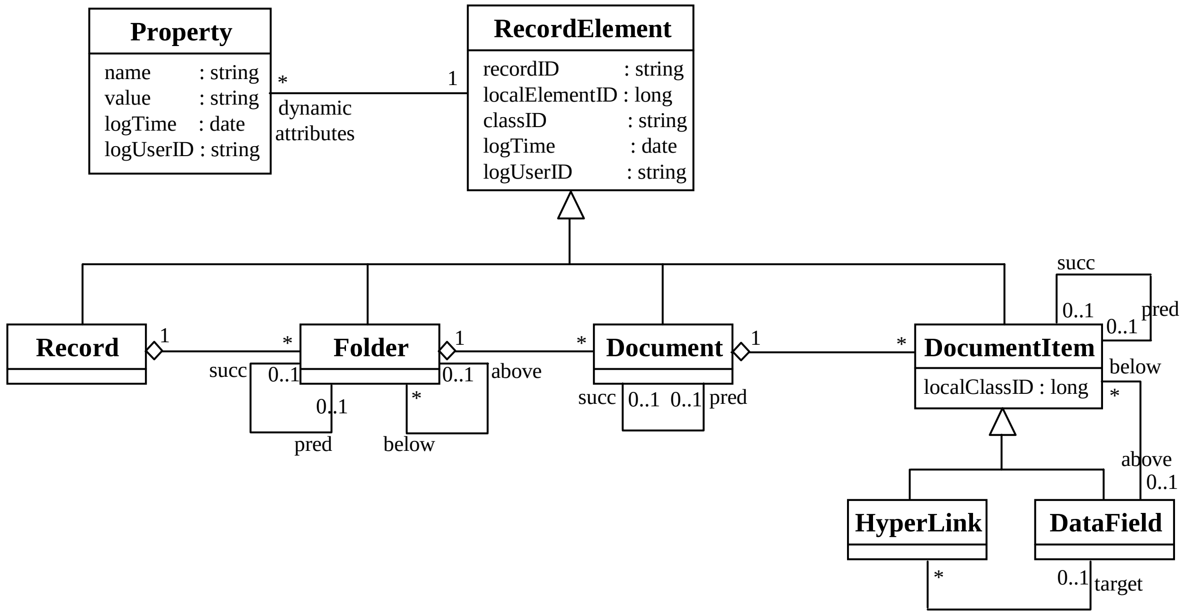
Example - An Electronic Patient Record (EPR) Server

Based on the results of *Synapses* - an EU project for the standardization of EPR's

Implementation Information Model



Interface Information Model



NR Implementation-Oriented Information Models (ImpIM)

Goals: a) to represent the information of interest, and b) to assure consistency in this information

Assure Consistency - Elementary Associations - Normalization

- Find *elementary associations* -
  associations that are sufficiently small to avoid the "repetition of information" and "the inability to represent certain information" problems, but not so small that they imply a "loss of information".
- Handled by a *normalization process* - but the technical details of normalization theory is not a prerequisite for good modelling.

Avoid Redundancy - Derivable Associations - Pragmatic considerations

- *Derivable associations* should be "read-only" and computed on demand to avoid redundancy that can lead to inconsistencies
- In practice not always possible - the essence is to be aware of it

Constraints and Business Rules

- Constraints are equally important to associations in defining which information can be represented
- What are derivable associations depends upon the business rules

Change Control

- Changes in ImpIM are often expensive
- ImpIMs are often made general and generic to better support changes
- It is easier to add, change or remove business rules than to change the association structure.

Performance and Platform Oriented

- ImpIM focuses on achieving an efficient and flexible implementation - thus influenced by performance issues, e.g. relating to the implementation platform

Interface Information Models (IntIM)

Goal: provide a description and documentation of how to use a set of related software components

A set of related components should always be accompanied by a corresponding IntIM

Change Control

IntIM is part of the *contract* between the components and their clients
Changing them is a "paper excercise", but clients are affected

IntIM can be more domain specific

There is no benefit in making an IntIM more general or generic, as when defining ImpIM's

Constraints and Business Rules

Does not concern consistency - only how a client can work with the components

Maximize Encapsulation

An IntIM does not concern how component interfaces are implemented - there need not be any correspondence between the IntIM and the ImpIM

Confine the effects of implementation changes
- The design of component interfaces should not reveal how they are implemented
- Focus on *what* an object offers to its clients, *not how* it does this

Documentation

E.g. IntIM may well describe detailed function signatures for documentation purposes

Interface Information Models (cont.)

Consistency and Redundancy

- Avoiding redundancy and distinguishing derivable versus non-derivable associations is irrelevant to IntIM.

  Redundancy in an IntIM can do no harm as long as consistency is maintained by the implementation

- Avoiding redundancy in an ImpIM implies that every "piece" of information is stored in one place, not duplicated several places.

  For every constraint that apply to an ImpIM there should be as *few* objects as possible, preferably just a single object, in charge of testing or maintaining this constraint.

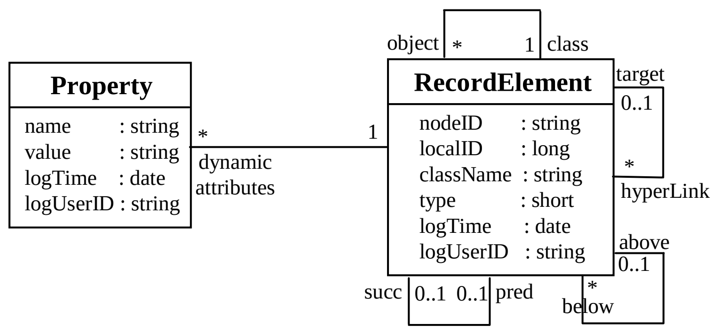
  This is not an issue for IntIM where the same information, or the same functionality, can be offered several places without introducing redundancy, inconsistencies, or hamper maintenance.

Interface Information Models (cont.)

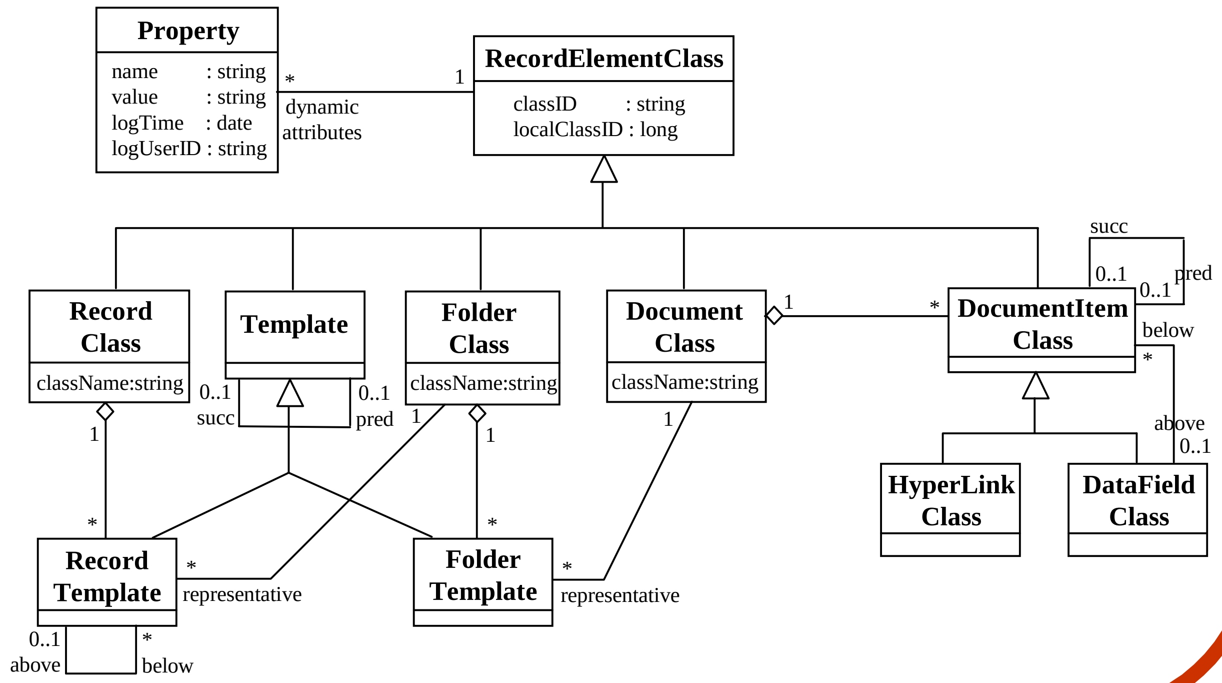
Connected Models and Logical Views

- ImpIM are connected models -
  or else models of independent systems
- Several IntIM can offer different logical views
  to the same system

Implementation Information Model



Record Class View Interface Information Model



Summary

We can distinguish two different kinds of information models for the design, implementation and documentation of sets of related software components.

Implementation Information Models (ImpIM)

- Used as a basis for implementing components and their interfaces
- Primary concern is to assure consistency, achieve good performance, and being flexible w.r.t. future changes

Interface Information Models (IntIM)

- An implementation-independent model that describe the components as perceived by clients using their interfaces
- Primary concern is conceptually simple (more domain specific), easy to use client interfaces with proper encapsulation such that technical, domain independent implementation changes are confined without affecting the interfaces and thus clients.

Similarities

They should both be the results of an *analysis/design* phase
- for ImpIM to understand and design an implementation, and
- for IntIM to understand and design good client interfaces

They can both be described by the same notation, but
- an ImpIM may not be considered a good IntIM since it is too implementation-oriented, too generic, or too awkward to use, and
- an IntIM may not be considered a good ImpIM by being too specific and thus not good for handling changes