Information Models for Component Design, Implementation and Documentation

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

Information Modeling

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.

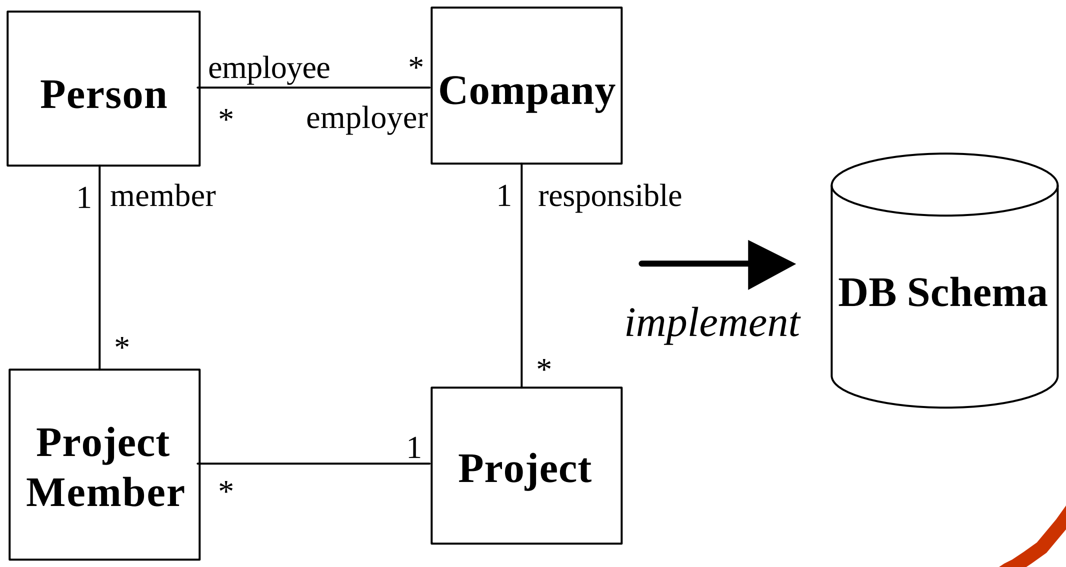
This to avoid that implementation-oriented issues (at least to a lesser degree) "clutters up" the conceptual view provided to clients working with these components.

Information Modelling

- a well-known topic (ER published '76)
- 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.

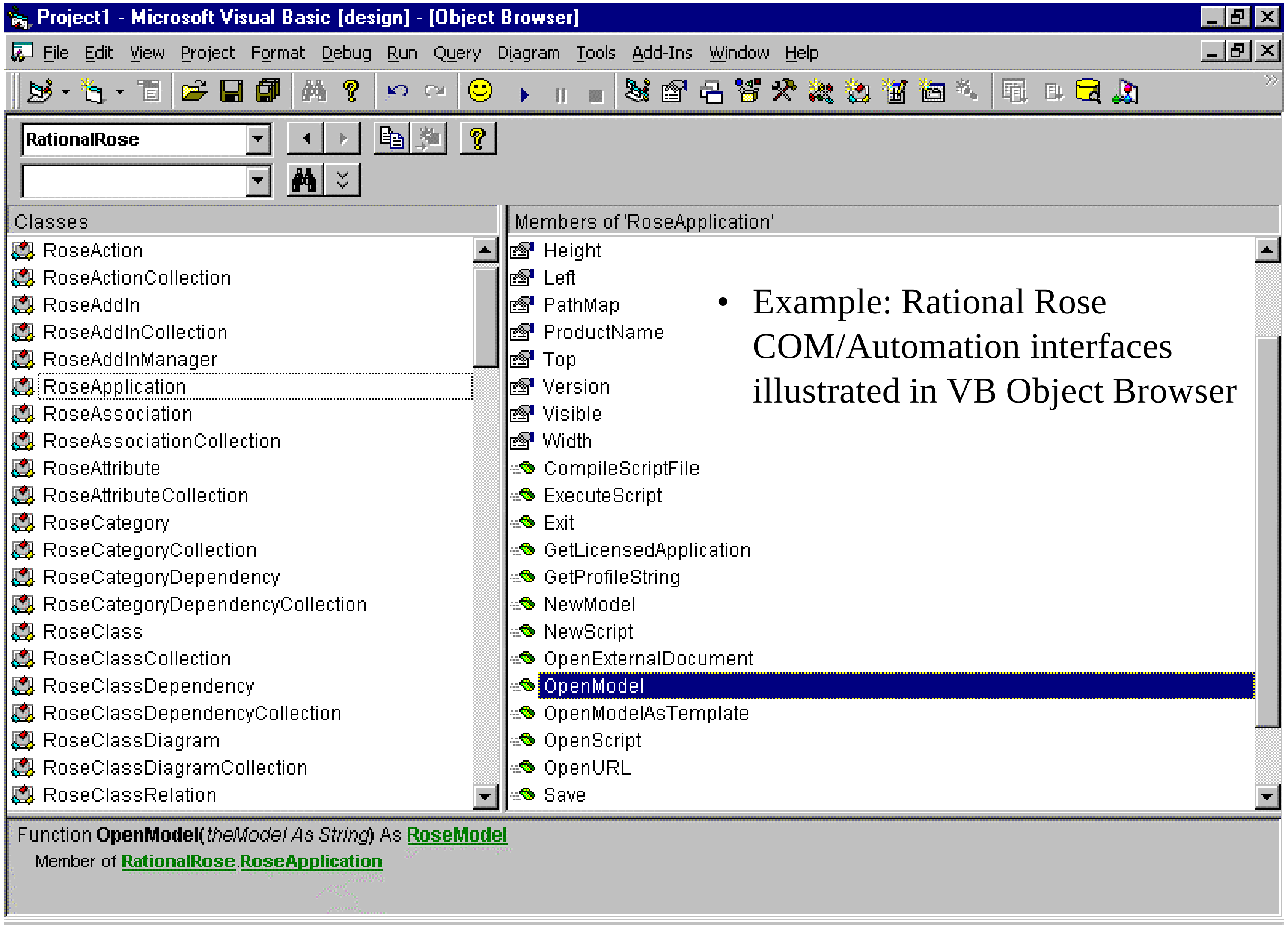
"Traditional" Information Modeling

- 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

- RoseAction
- RoseActionCollection
- RoseAddIn
- RoseAddInCollection
- RoseAddInManager
- RoseApplication
- RoseAssociation
- RoseAssociationCollection
- RoseAttribute
- RoseAttributeCollection
- RoseCategory
- RoseCategoryCollection
- RoseCategoryDependency
- RoseCategoryDependencyCollection
- RoseClass
- RoseClassCollection
- RoseClassDependency
- RoseClassDependencyCollection
- RoseClassDiagram
- RoseClassDiagramCollection
- RoseClassRelation

Members of 'RoseApplication'

- Height
- Left
- PathMap
- ProductName
- Top
- Version
- Visible
- Width
- CompileScriptFile
- ExecuteScript
- Exit
- GetLicensedApplication
- GetProfileString
- NewModel
- NewScript
- OpenExternalDocument
- OpenModel
- OpenModelAsTemplate
- OpenScript
- OpenURL
- 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

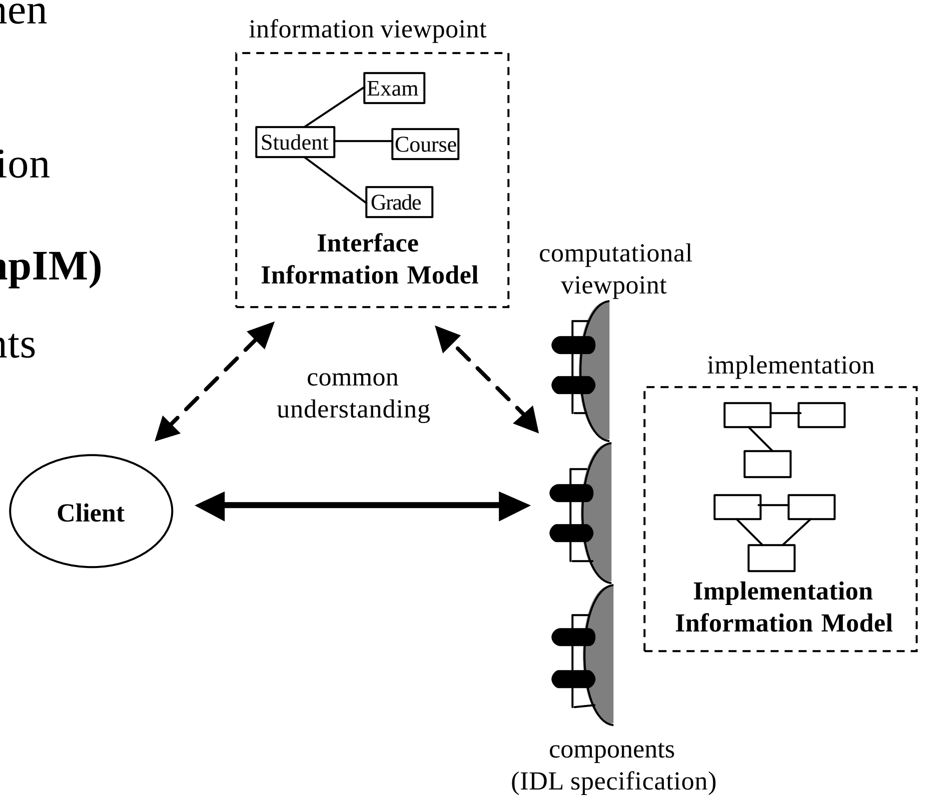
Two different kinds of IM for component-oriented systems where component implementations are encapsulated behind interfaces of functions offered to their clients

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)

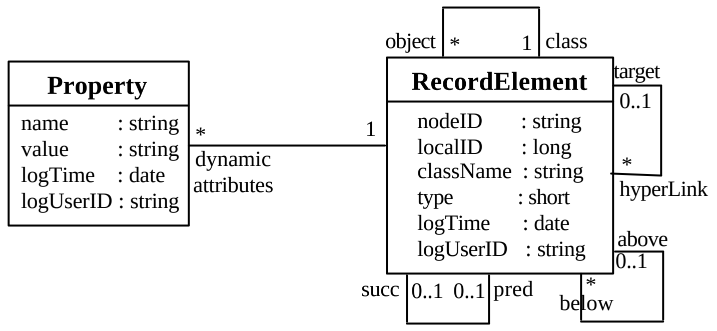
A basis for implementing the components



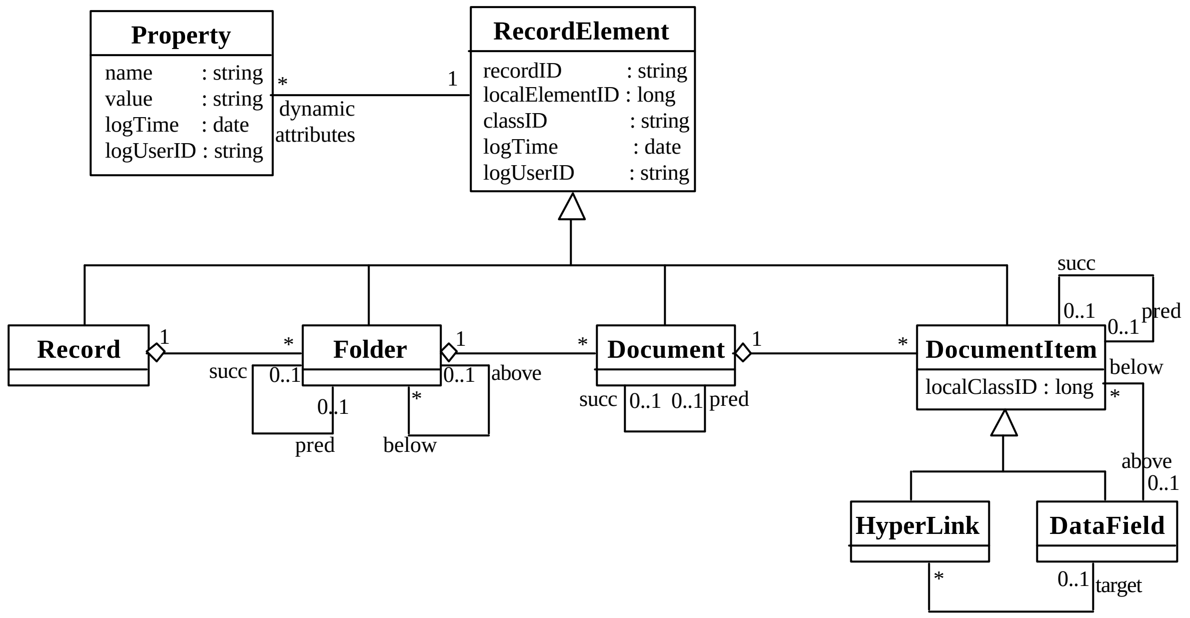
Example - An Electronic Patient Record (EPR) Server

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

Implementation Information Model



Interface Information Model



Implementation-Oriented Information Models

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

Relational vs Object-Oriented Information Models

Relational IM: ER-like diagrams, NIAM
Object-Oriented IM: UML Class Diagram, OMT Object Model

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

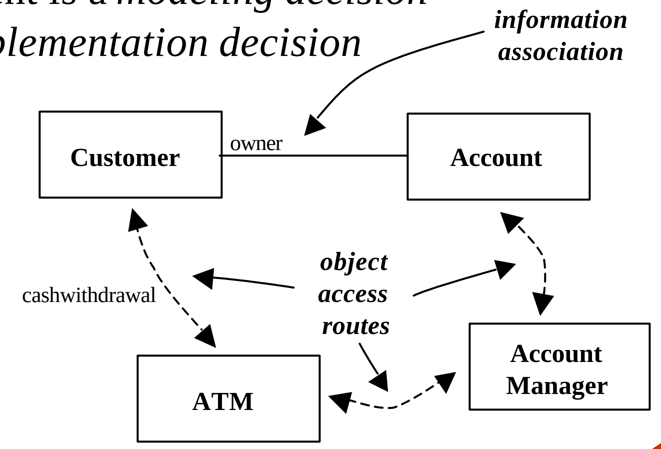
Object Information Associations versus Access Routes

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.



Interface Information Models

Interface Information Models (IntIM) are made to 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.

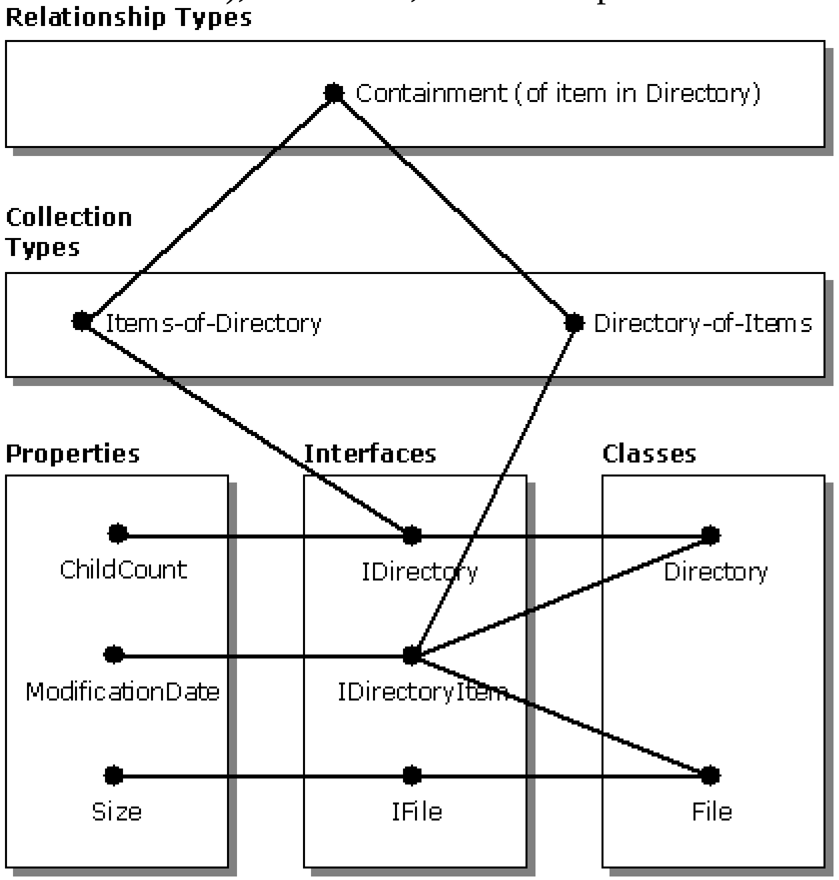
  For every constraint that apply to an ImpIM there should be as *few* objects as possible, preferably just a single object, in charge of testing or maintaining this constraint.

  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.

Implementation and Interface Information Models of Microsoft Repository

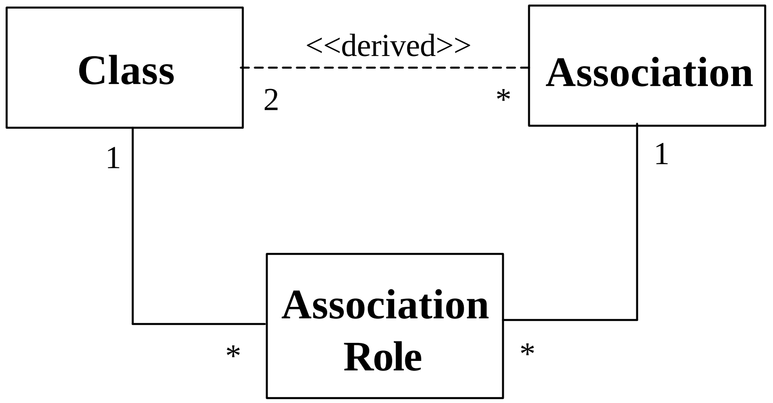
RTIM - Repository Type Information Model

- A domain independent information model
- Made to record and retrievemeta-information on a variety of domains (e.g. UML, DB Schemas, Components, Datatypes, and more)
- Basic concepts: Class, Interface, Property (attribute and method), Collection, Relationship



Repository UML Information Model

- A domain dependent information model
- Made to record and retrieve meta-information on UML models (e.g. fromRational Rose models)
- Accompanies COM/Automation interfaces
- Implemented by RTIM
- Currently “too normalized” - should allow for “redundancy”; e.g. Class-Association-Role relationships



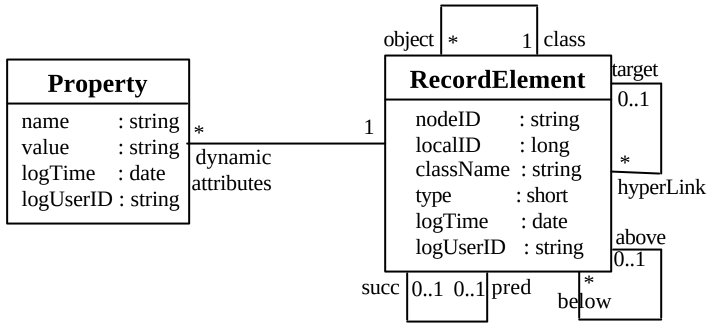
```
IClass ==
    GetRoles()->IAssociationRoleColl
  + [GetAssociations()->IAssociationColl]
IAssociation ==
    GetRoles()->IAssociationRoleColl
  + [GetClasses()->IClassColl]
IAssociationRole ==
    GetAssociation()->IAssociation
    GetClass()->IClass
```

Interface Information Models (cont.)

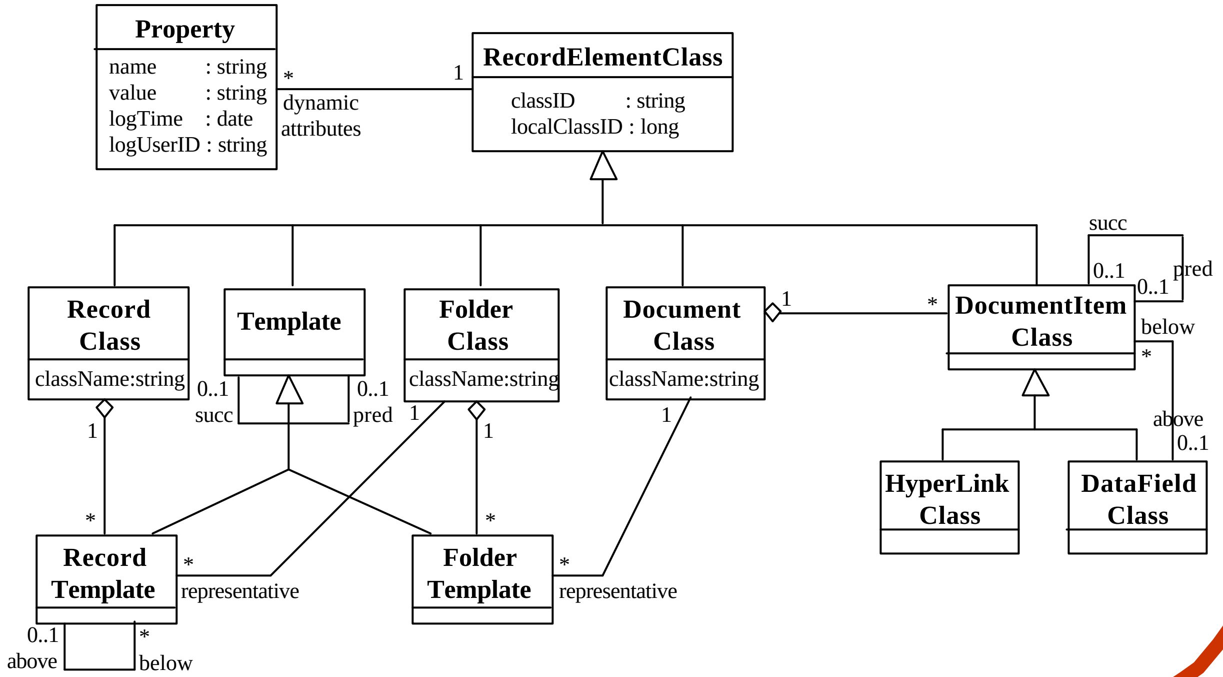
Connected Models and Logical Views

- 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