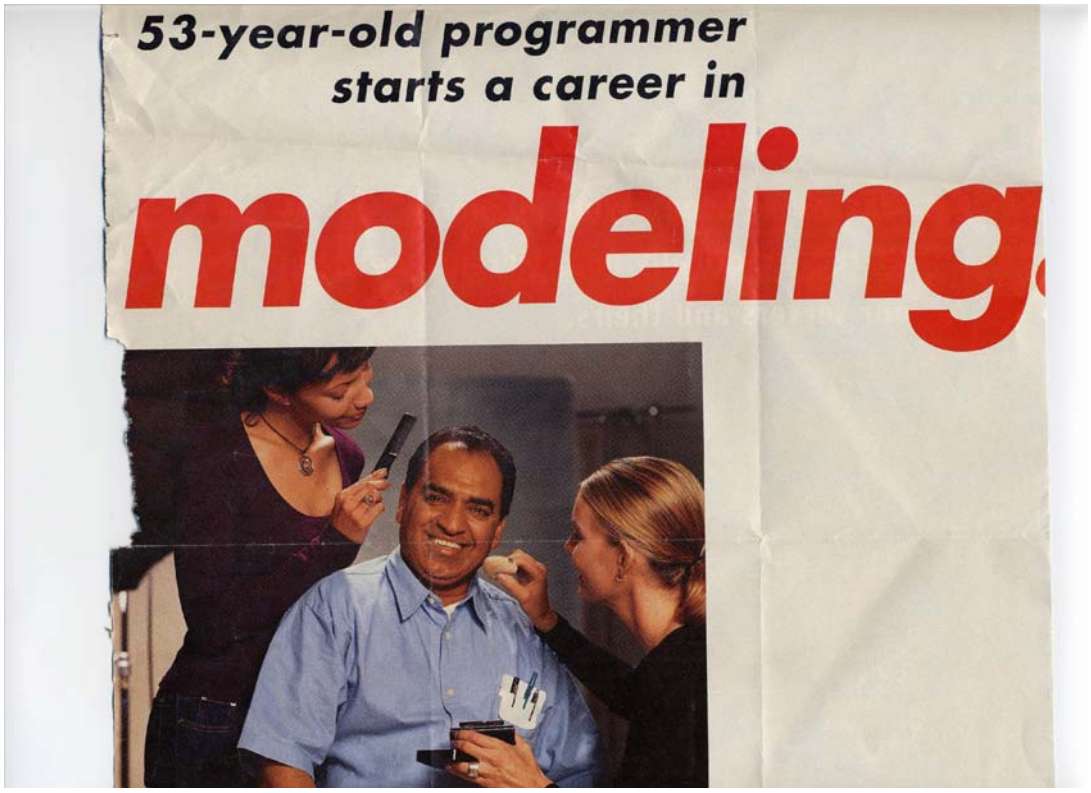
Understanding the UML

Michael R. Blaha
Modelsoft Consulting Corp
E-mail: blaha@computer.org
www.superdataguy.com

June 11, 2014



What is the UML?

The UML (Unified Modeling Language) is a graphical language for modeling software.

The UML has a variety of diagrams. One of these diagrams (***the class model***) is relevant for databases.

Why Use the UML? What Is It Good For?

The UML class model has advantages for data models.

- Better communication with business experts.
  - The UML avoids confusing database details.
    - Can show attributes without addressing keys.
    - Avoids distinction between independent and dependent entities.
    - Can show role names without showing entity FKs.
    - Can show selected relationships by package / subject area.
- Better communication with programmers.
  - Most programmers are familiar with the UML.
- Support for abstraction.
  - It is helpful to suppress database details when thinking about deep concepts such as data modeling patterns.

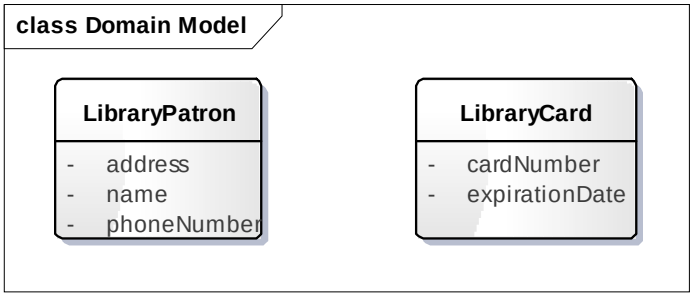
What Are Weaknesses of the UML?

The UML class model does not address database details.

- Lack of support for database design.
  - The UML notation does not cover database design.
  - Most UML tools also lack database design support.
    - Enterprise Architect has some database support.
- Overemphasis on programming jargon.
  - The UML creators focused on programming and ignored databases.
  - Ironically the programming jargon is superficial and the UML has much to offer for database applications.

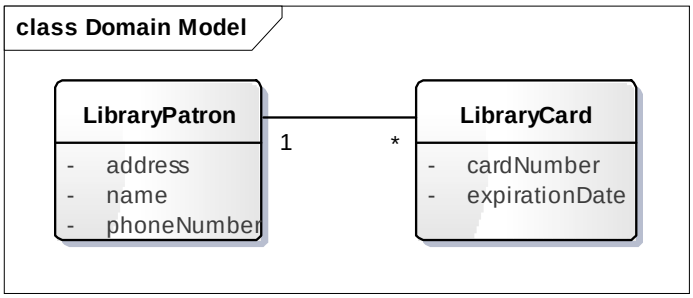
Basic Construct: Class

- ***Object*** — a concept, abstraction, or thing with identity and application meaning.
- ***Class*** — a description of similar objects.
  - **UML notation**. A box with the class name at the top.
- ***Value*** — a piece of data for an object.
- ***Attribute*** — a description of similar values.
  - Analogy — class::object as attribute::value.
  - **UML notation**. In the second portion of the class box.



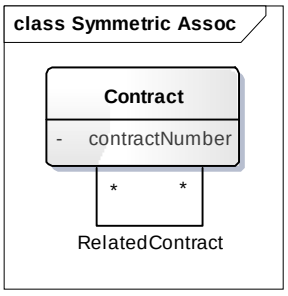
Basic Construct: Association

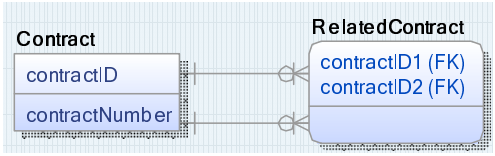
- ***Link*** — a physical or conceptual connection among objects.
- ***Association*** — a description of similar links.
  - **UML notation**. A line connecting the related classes.
- ***Association end*** — an alias for a foreign key.
  - **UML notation**. A legend next to the class–association intersection.
- ***Multiplicity*** — the number of occurrences of one class that may relate to a single occurrence of an associated class.
  - **UML notation**. Usually “1”, “0..1”, and “*” (“many” — zero or more).



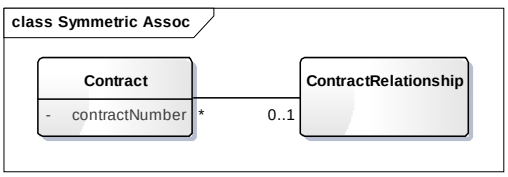
Basic Construct: Association
Symmetric Association

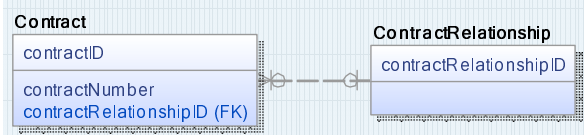
- The following UML model has an implementation problem.





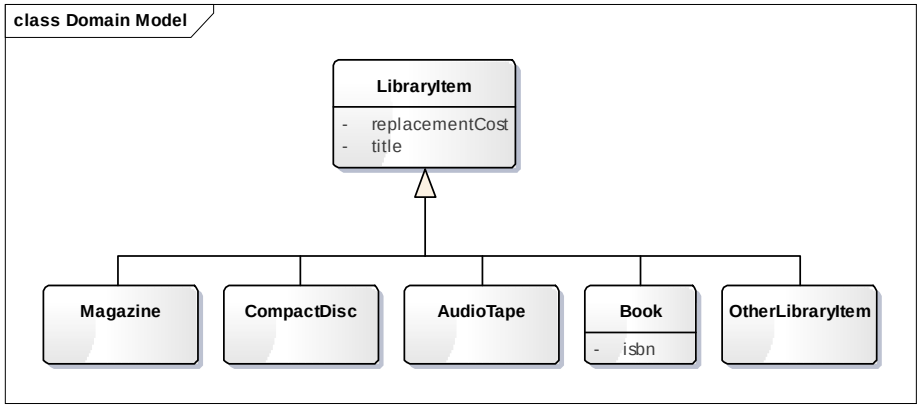
- Solution: restate the model.





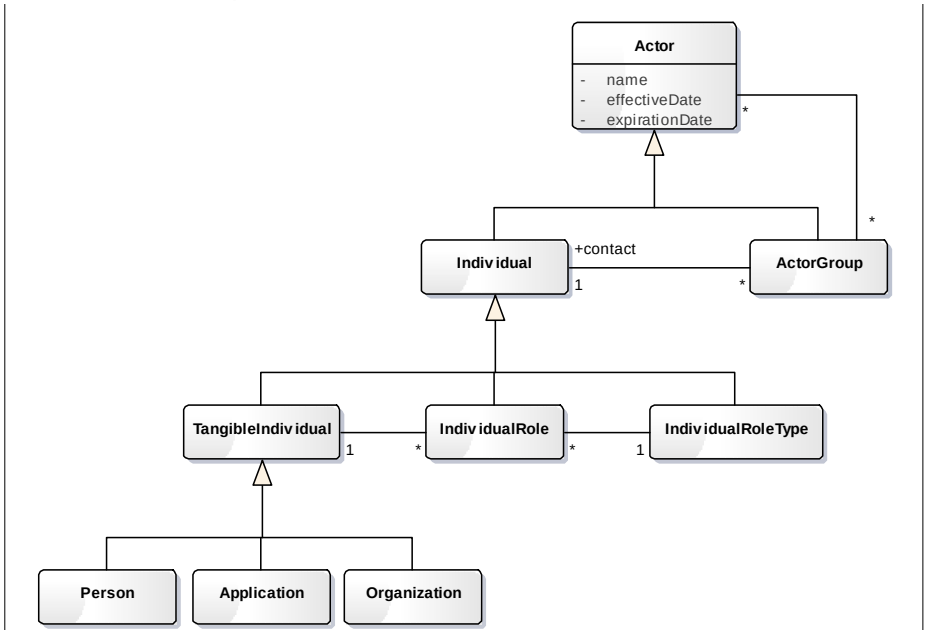
Basic Construct: Generalization

- ***Generalization*** — couples a class (the ***superclass***) to one or more variations of the class (the ***subclasses***).
  - **UML notation**. A large hollow arrowhead points to the superclass. Lines fan out towards the subclasses.
  - The superclass has common data.
  - The subclasses have specific data.



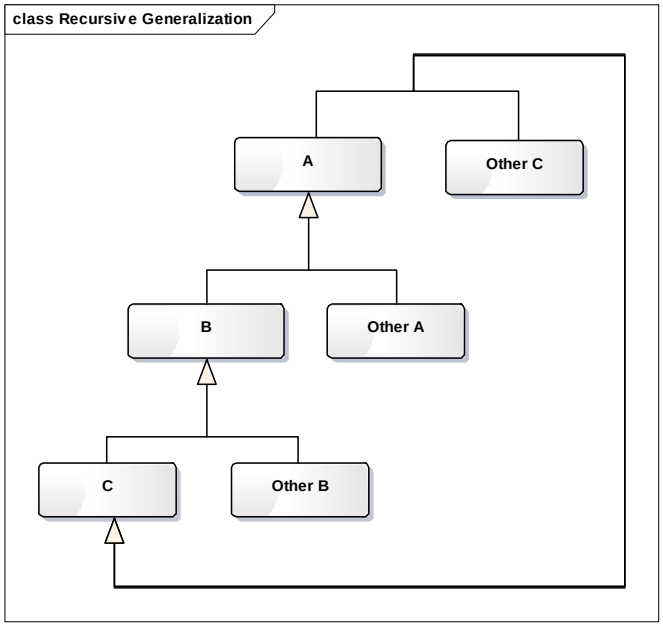
Basic Construct: Generalization Actor Archetype

- There can be multiple generalization levels.



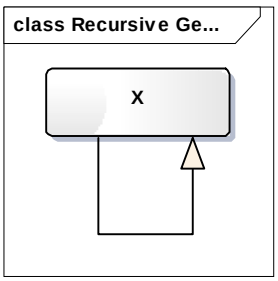
Basic Construct: Generalization
Missing Tool Support

- Some UML tools do not fully support generalization. (ERwin is OK.)

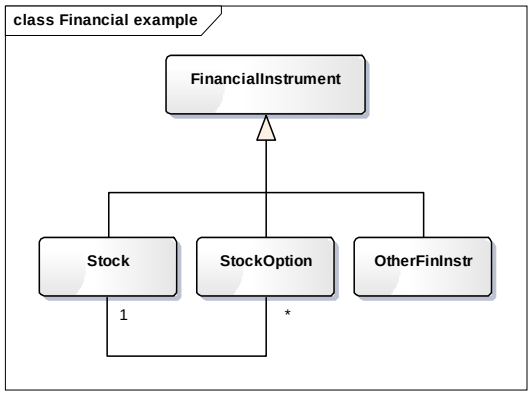


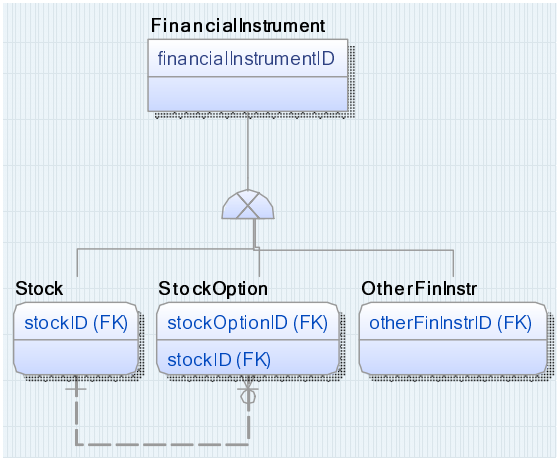
Basic Construct: Generalization
Missing Tool Support

- Some UML tools do not fully support generalization. (ERwin is OK.)



Basic Construct: Generalization
Propagating Referential Integrity



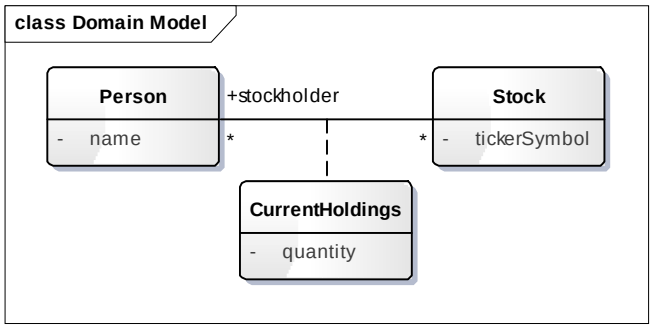


Basic Construct: Generalization
Propagating Referential Integrity (cont.)

```
CREATE TABLE FinancialInstrument
( fiID       bigint identity(1,1) NOT NULL ,
  CONSTRAINT FI_PK PRIMARY KEY (fiID) )

CREATE TABLE OtherFinInstr
( ofiID      bigint NOT NULL ,
  CONSTRAINT OFI_PK PRIMARY KEY (ofiID) )

CREATE TABLE Stock
( stockID    bigint NOT NULL ,
  CONSTRAINT Stock_PK PRIMARY KEY (stockID) )

CREATE TABLE StockOption
( soID       bigint NOT NULL ,
  stockID    bigint NOT NULL ,
  CONSTRAINT SO_PK PRIMARY KEY (soID) )

CREATE NONCLUSTERED INDEX SO_i1 ON StockOption ( stockID )

ALTER TABLE OtherFinInstr ADD CONSTRAINT OFI_FK1 FOREIGN KEY (ofiID)
  REFERENCES FinancialInstrument ON DELETE CASCADE

ALTER TABLE Stock ADD CONSTRAINT Stock_FK1 FOREIGN KEY (stockID)
  REFERENCES FinancialInstrument ON DELETE CASCADE

ALTER TABLE StockOption ADD CONSTRAINT SO_FK1 FOREIGN KEY (soID)
  REFERENCES FinancialInstrument ON DELETE CASCADE

ALTER TABLE StockOption ADD CONSTRAINT SO_FK2 FOREIGN KEY (stockID)
  REFERENCES Stock ON DELETE CASCADE
```

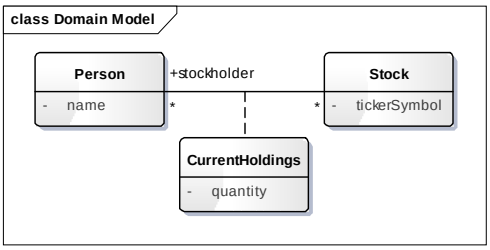
Advanced Construct: Association Class

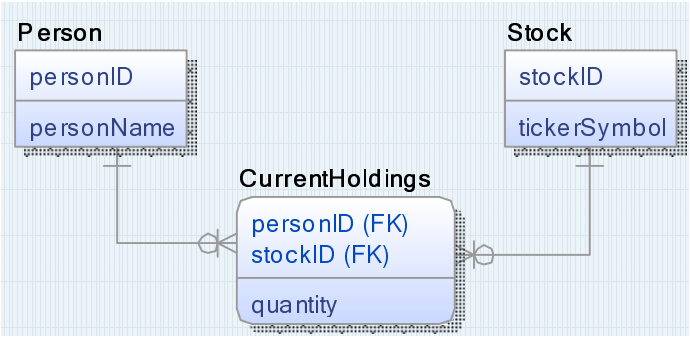
- ***Association class*** — an association that is also a class.
  - **UML notation**. A box connected to the association with a dotted line.
  - Like an association, an association class has dependent identity.
  - Like a class, an association class can have attributes, operations, and associations.



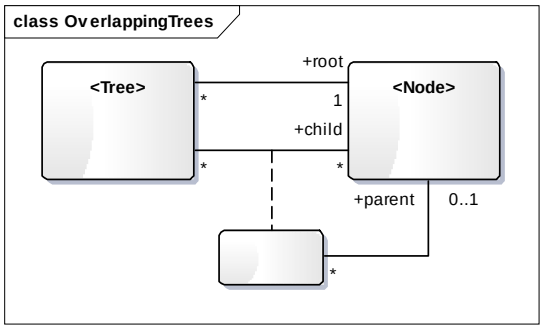
Advanced Construct: Association Class (cont.)

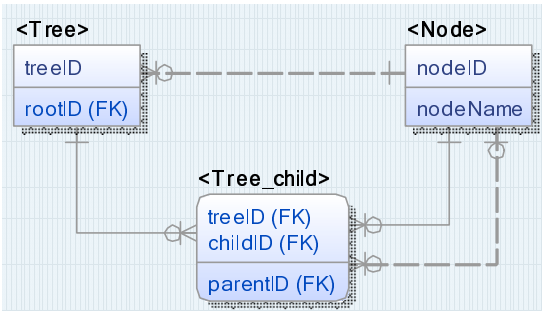
- ***Association class***...





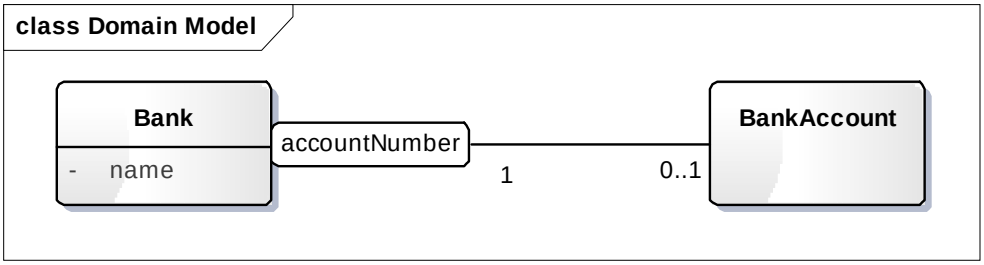
Advanced Construct: Association Class Overlapping Trees Template





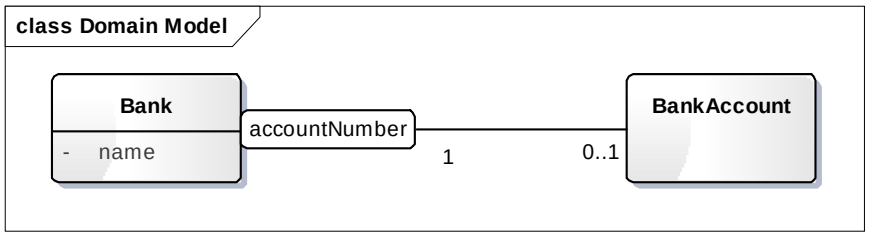
Advanced Construct: Qualified Association

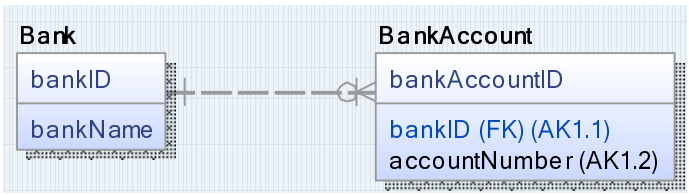
- ***Qualified association*** — a qualifier attribute disambiguates objects for a "many" assoc end.
  - **UML notation**. A small box on the association line by the source class. The source class plus the qualifier yields the target class.
  - The qualifier selects among the target objects, reducing the effective multiplicity, often from "many" to "one".
  - Associations that are x-to-many can have qualifiers.



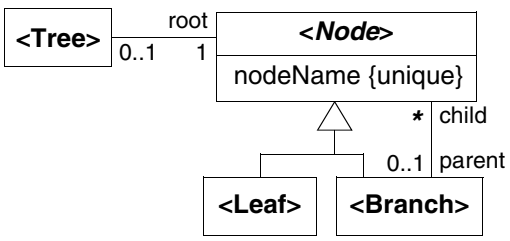
Advanced Construct: Qualified Assoc (cont.)

- ***Qualified association***…

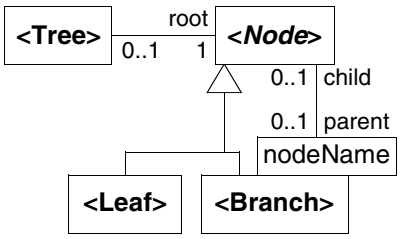




Advanced Construct: Qualified Association Structured Tree, UML

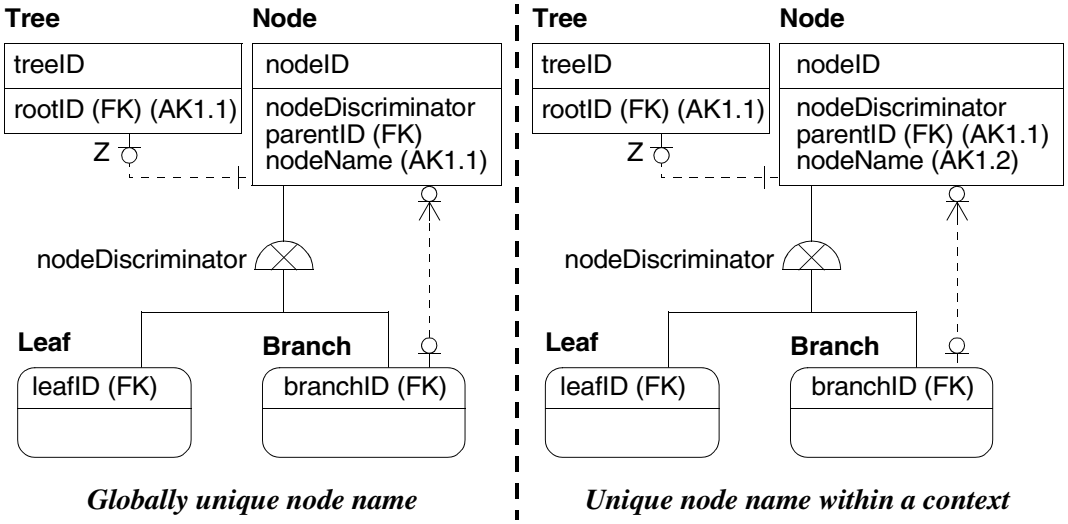


Globally unique node name



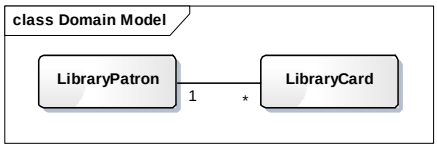
Unique node name within a context

Advanced Construct: Qualified Association Structured Tree, IE

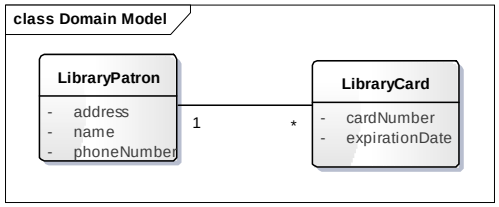


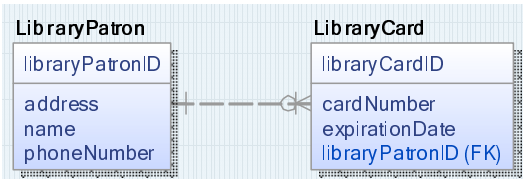
UML vs. IE — Conceptual, Logical, Physical Models

- ***Conceptual model***

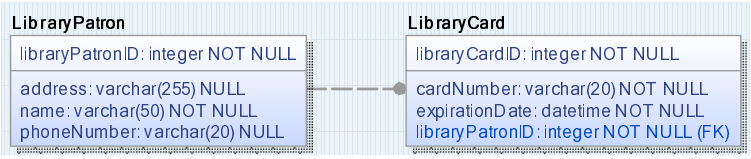


- ***Logical model***





- ***Physical model***



UML vs. IE
Operational, Analytical, Enterprise Models

- UW Health — analytical, a giant medical data warehouse.
  - Use IE. The UML provides no benefit for a star schema.
- Horse racing model — operational
  - Use UML. The specifier was building the application. The UML was convenient for his programming.
- Avelo — operational, enterprise
  - Use UML. They are a UML shop. They have a high skill level with Enterprise Architect and generate database schema from it.
- Caesars — operational
  - Use UML. It was understandable to business staff and programmers.
- Caesars — enterprise
  - Use IE. ERwin's data dictionary reports are superb. Dealing with identity is a hassle.

UML and Relational Databases

Forward engineering — generally favor UML

- Operational — use a blend of two notations.
  - UML for conceptual and logical model.
  - IE for physical model.
- Analytical — just use IE
- Enterprise — prefer UML

Reverse engineering — generally favor IE

- ERwin has automation for loading schema.
- ERwin has nice layout features that are helpful for rapid reverse engineering.

UML and XSD

The current SOA practice does not pay attention to data modeling.

- Services provide functionality.
  XSD files specify the interface.
- XSD editors are used for specifying individual XSD files.
- Nothing is being used for coordinating XSD files.
- This is a real problem when you have hundreds of services that store and retrieve data (a common situation).

Forward engineering

- There are approximate rules for mapping UML to XSD.
- The literature is incomplete.

Reverse engineering

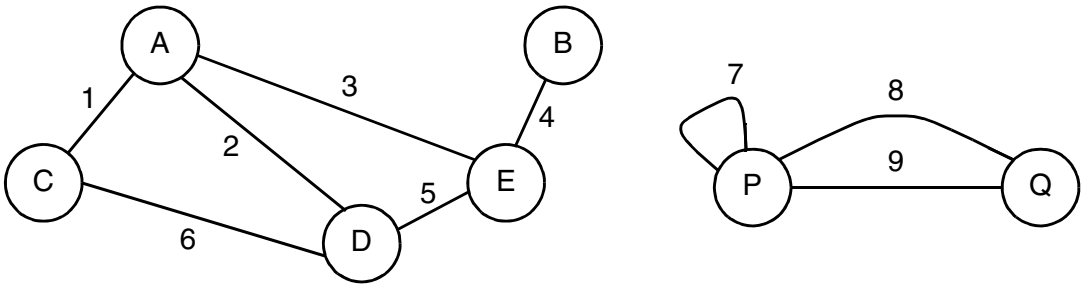
- There is a lack of rules for mapping XSD to UML.
- The literature is sparse.
- XSD is a messy language.

Model Quality

- Model quality can be the difference between the success or failure of an application.
- Here are several techniques for assessing and improving model quality.
  - Normal forms.
  - Database constraints.
  - Hillard's graph complexity.
  - Hoberman's data model scorecard®.
- I also pay attention to the size of a data model (the number of tables).
  - The larger the data model, the more development work.

Hillard's Graph Complexity

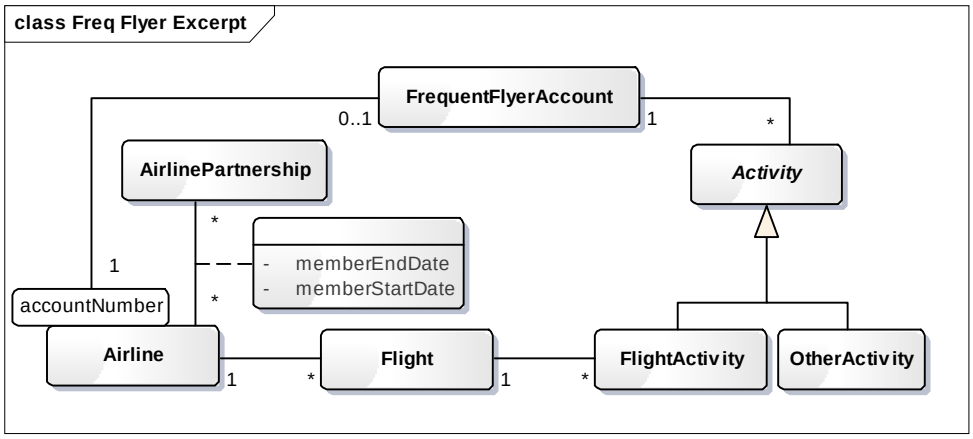
- Hillard assesses the quality of a model by equating a data model to an undirected graph.
  - A ***node*** is a class or many-to-many association.
  - An ***edge*** is a one-to-x association or a superclass-subclass coupling.



Hillard's Graph Complexity (cont.)

- Aspects of complexity.
  - **Traversal length**. Long traversals can be difficult to understand, error prone, and costly to develop.
  - **Number of edges connected to a node**. Nodes with three or more edges offer choices for traversal.
- Hillard computes the following graph metrics:
  - **Average degree**. The average number of edges per node.
  - **Geodesic distance**. The minimum number of edges needed to traverse between a pair of nodes.
  - **Average geodesic distance**. The average of the geodesic distance for all node pairs.

Hillard's Complexity Example



- Avg degree: 2.0
  - OtherActivity–1, FlightActivity–2, Activity–3
  - FrequentFlyerAccount–2, Flight–2
  - AirlinePartnership–1, Airline–3, many-to-many–2

Hillard Complexity Example (cont.)

- Average geodesic distance: 2.2

Geodesic distance

	O	FA	Act	FFA	F	AP	Air	AMM
OtherActivity	XX	2	1	2	3	5	3	4
FlightActivity	2	XX	1	2	1	4	2	3
Activity	1	1	XX	1	2	4	2	3
FreqFlyerAcct	2	2	1	XX	2	3	1	2
Flight	3	1	2	2	XX	3	1	2
AirlinePart	5	4	4	3	3	XX	2	1
Airline	3	2	2	1	1	2	XX	1
M-to-M assoc	4	3	3	2	2	1	1	XX

Tools

UML. Did a UML triage ten years ago and looked at 42 tools.

- EA — Excellent tool. Cost ~$200.
- Magic Draw — Excellent tool. Cost ~$200.
- RSA — Excellent tool. Integrates multiple models. Cost ~$5000.
- Rational Rose — Excellent tool. Cost ~$2000. Obsolete.

IE

- ERwin — Excellent tool. Cost ~$5000.
- Dezign — Credible tool, but not as good as ERwin. Cost ~$200.

Documenting Data Models

- Data dictionary
  - Supported by most tools.
  - Convenient for data modelers.
  - Overwhelming for the business reader.
- Narrative
  - Write prose explaining the model and intersperse diagrams.
  - A pain to keep consistent with model revisions.
  - Friendly for the business reader.

In any case, it is important to define terminology for models.

Presenting Data Models

- Walk through the diagrams using a modeling tool.
  - Seldom effective. Disjointed presentation.
- Slides.
  - Show diagrams for important portions of the model.
  - Define terminology.
  - Walk through the model.
- Create a “spontaneous” model.
  - Create the model live, on the fly, in front of the audience.
  - Often use a previously constructed model as a guide.
  - Tell the audience that there is already an existing model and on-the-fly construction is just a presentation artifact.

(Sometimes I do construct a model from scratch, live on-the-fly.)

UML and the Zachman Framework

- Why — not covered by the UML.
- How — use cases.
- What — the UML class model.
  - The conceptual model can be the UML model instead of an entity relationship model.
  - The UML really is just an ER model. ER is a family of similar notations.
- Who — use cases.
- Where — not covered by the UML.
- When — state diagrams.