From UML to Relations

Carla Teixeira Lopes

Bases de Dados
Mestrado Integrado em Engenharia Informática e Computação, FEUP

Based on Jennifer Widom slides

UML key concepts

Classes

Associations

Association Classes

Generalizations

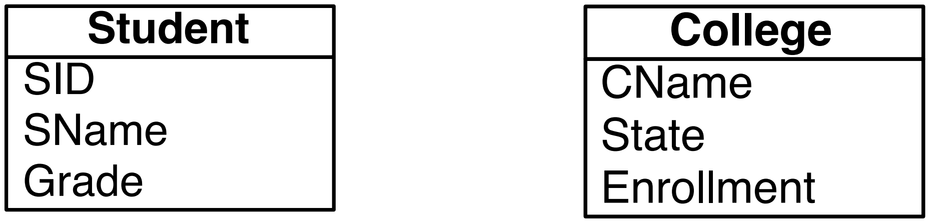
Composition & Aggregation

Constraints

Derived Elements

Classes

Every class becomes a relation



Student
SID
SName
Grade

College
CName
State
Enrollment

Student (SID, SName, Grade)

College (CName, State, Enrollment)

UML key concepts

~~Classes~~

Associations

Association Classes

Generalizations

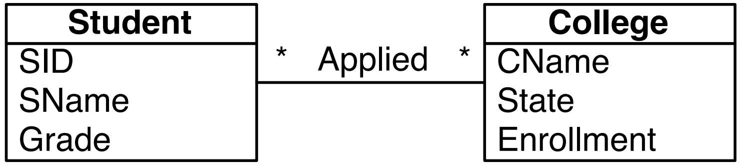
Composition & Aggregation

Constraints

Derived Elements

Many-to-many associations

Add a relation with key from each side



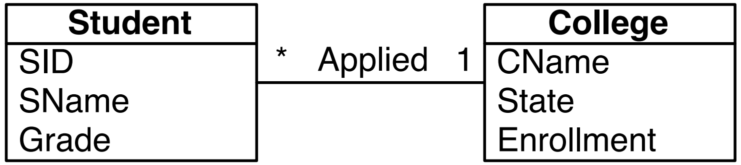
Student (<u>SID</u>, SName, Grade)

College (<u>CName</u>, State, Enrollment)

Applied (<u>SID</u>->Student, <u>CName</u>->College)

Many-to-one associations

Add a foreign key to the **many** side of the relationship to the relation in the one side



Student

SID
SName
Grade

* Applied 1

College

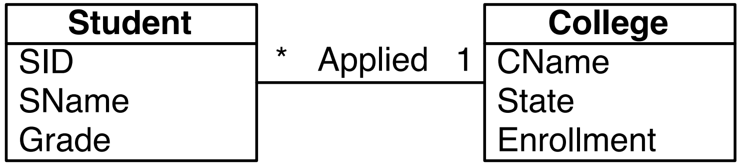
CName
State
Enrollment

Student (SID, SName, Grade, CName->College)

College (CName, State, Enrollment)

Many-to-one associations

Add a relation with key from the many side



Student
SID
SName
Grade

* Applied 1

College
CName
State
Enrollment

Student (SID, SName, Grade)

College (CName, State, Enrollment)

Applied (SID->Student, CName->College)

Many-to-one associations

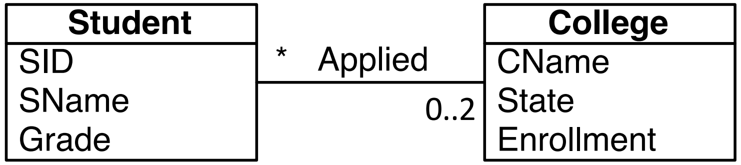
Add a foreign key to the many side of the relationship to the relation in the one side

- Most common
- Less relations in the schema
- Increased performance due to a smaller number of relations

Add a relation with key from the many side

- Increased rigour of the schema
- Increased extensibility

Question



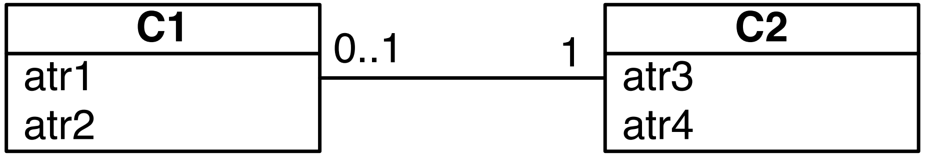
Suppose we had 0..2 on the right-hand side, so students can apply to up to 2 colleges. Is there still a way to "fold in" the association relation in this case, or must we have a separate Applied relation?

Yes there is a way

No, if it's not 0..1 or 1..1 then Applied is required

One-to-one associations

Add a foreign key from one of the relations to the other



C1 (<u>atr1</u>, atr2, c2_id->C2)

C2 (<u>atr3</u>, atr4)

Add the foreign key to the relation that is expected to have less tuples

Add a unique key constraint to the foreign key

UML key concepts

~~Classes~~

~~Associations~~

Association Classes

Generalizations

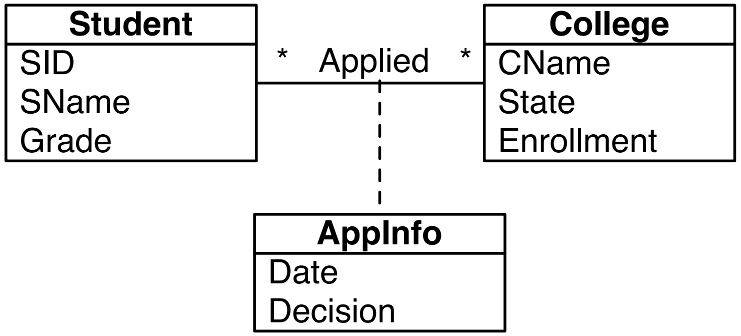
Composition & Aggregation

Constraints

Derived Elements

Association classes

Add attributes to relation for association



Student
SID
SName
Grade

* Applied *

College
CName
State
Enrollment

AppInfo
Date
Decision

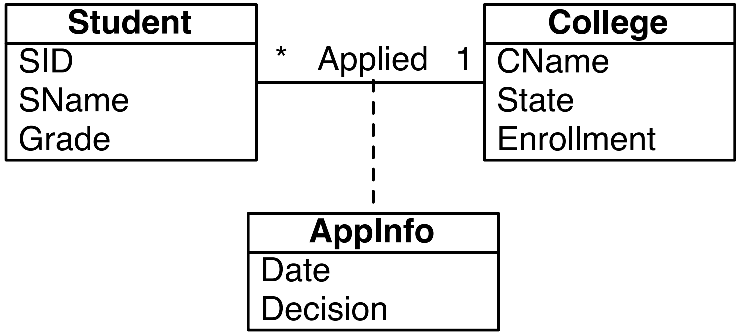
Student (SID, SName, Grade)

College (CName, State, Enrollment)

Applied (SID->Student, CName->College, Date, Decision)

Association classes

Add attributes to relation for association

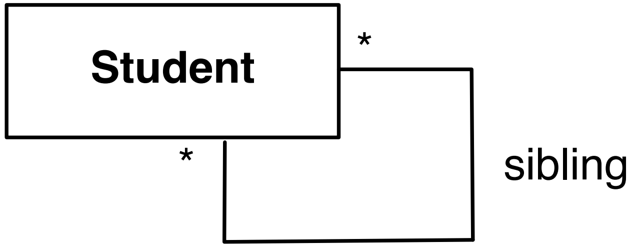


Student (<u>SID</u>, SName, Grade)

College (<u>CName</u>, State, Enrollment)

Applied (<u>SID</u>->Student, CName->College, Date, Decision)

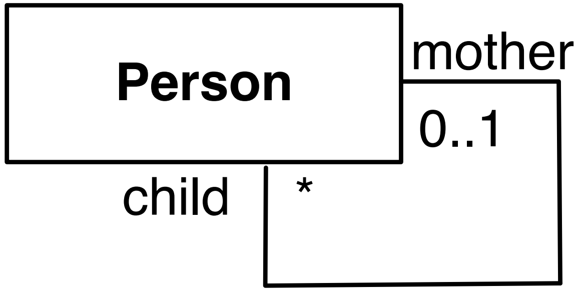
Self associations



Student (id, …)

Sibling (sid1->Student, sid2->Student)

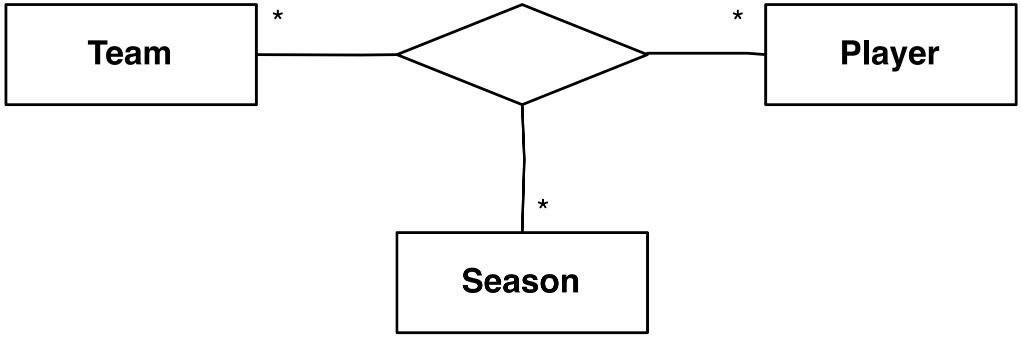
Self associations



Person (<u>id</u>, …)

Relationship (mother->Person, <u>child-</u>>Person)

Associations n-ary



Relation with key from each side

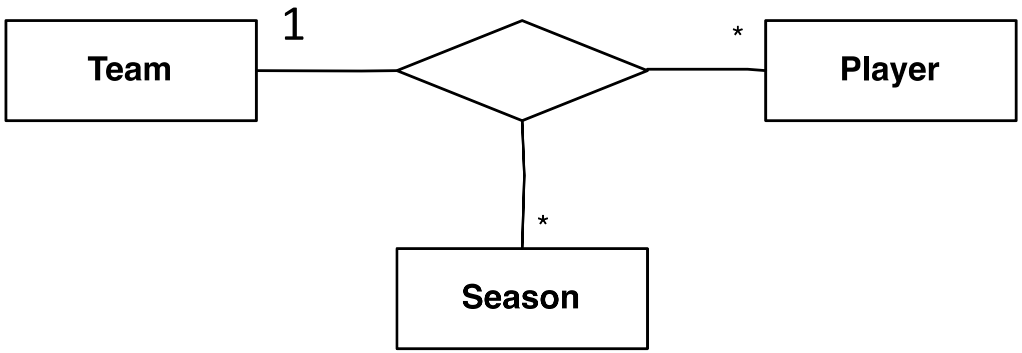
Team (ID, …)

Player (ID, …)

Season (ID, …)

PlayerSeasonTeam (PlayerID->Player, SeasonID->Season, TeamID->Team)

Associations n-ary



Relation with key from each side

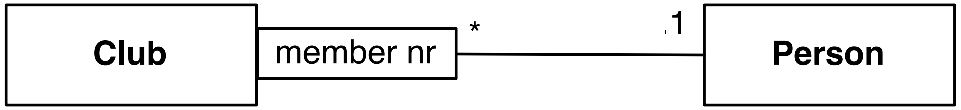
Team (ID, …)

Player (ID, …)

Season (ID, …)

PlayerSeasonTeam (PlayerID->Player, SeasonID->Season, TeamID->Team)

Qualified associations



Club (ClubID, …)

Person (PersonID, …)

Member (ClubID->Club, PersonID->Person, MemberNr)

{ClubID, MemberNr} UK

UML key concepts

~~Classes~~

~~Associations~~

~~Association Classes~~

Generalizations

Composition & Aggregation

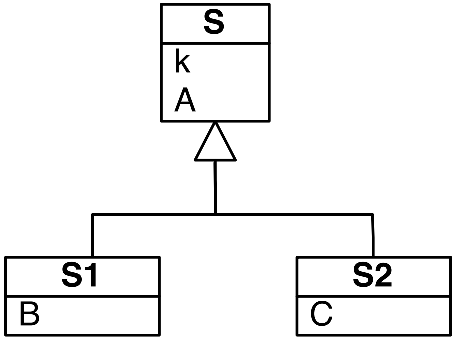
Constraints

Derived Elements

Generalizations

3 conversion strategies

- E/R style
- Object-oriented
- Use nulls



Best translation may depend on the properties of the generalization

E/R style

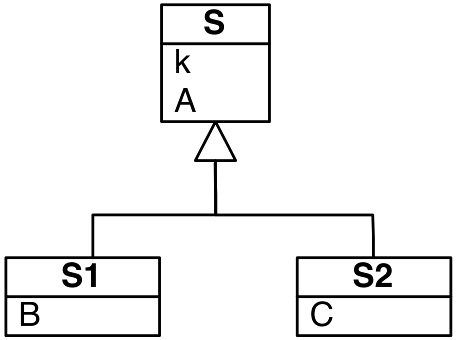
A relation per each class

Subclass relations contain superclass key + specialized attributes

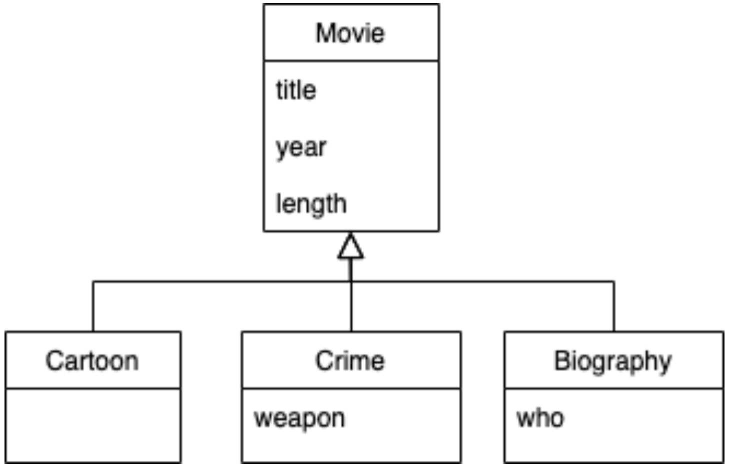
S(<u>k</u>, A)

S1(<u>k</u>->S, B)

S2(<u>k</u>->S, C)



E/R style



Movie (title, year, length)

Cartoon (title->Movie.title, year->Movie.year)

Crime (title->Movie.title, year->Movie.year, weapon)

Biography (title->Movie.title, year->Movie.year, who)

Object-oriented

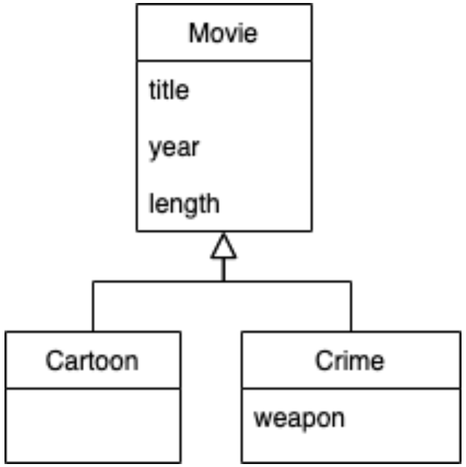
Create a relation for all possible subtrees of the hierarchy

- Schema has all the possible attributes in that subtree

In complete generalizations, the relation for the subtree with only the superclass may be eliminated

Object-oriented because each object belongs to one and only one subtree

Object-oriented in overlapping generalizations



Movie (title, year, length) → Not necessary if generalization is complete

MovieCartoon (title, year, length)

MovieCrime (title, year, length, weapon)

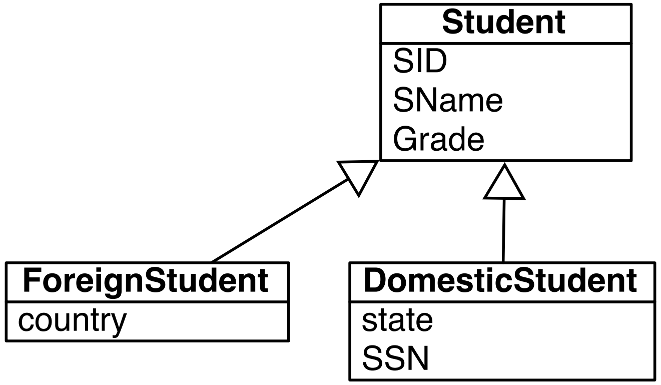
MovieCartoonCrime (title, year, length, weapon)

Object-oriented in disjoint generalizations

Student (SID, SName, Grade)

ForeignStudent (SID, SName, Grade, country)

DomesticStudent (SID, SName, Grade, state, SSN)



Or, if it is complete:

ForeignStudent (SID, SName, Grade, country)

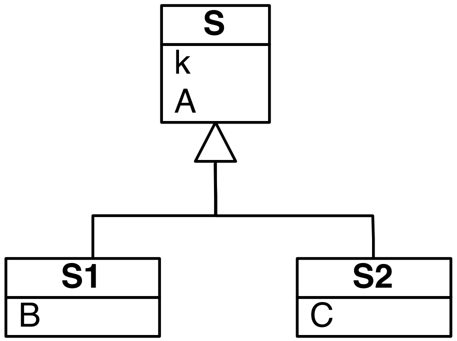
DomesticStudent (SID, SName, Grade, state, SSN)

Use nulls

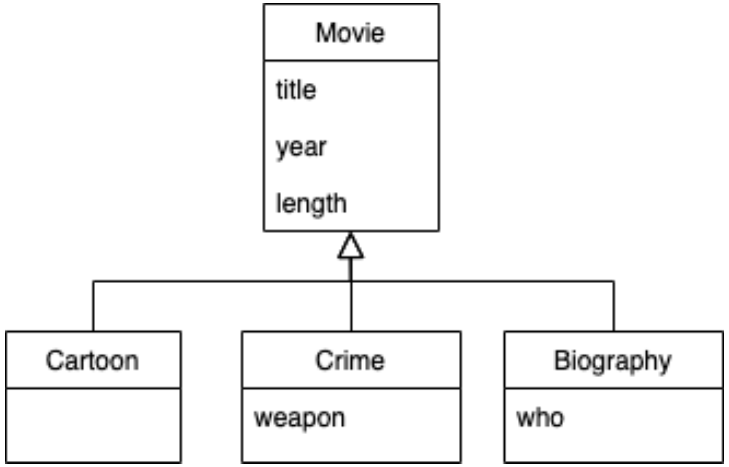
One relation with all the attributes of all the classes

NULL values on non-existing attributes for a specific object

S(k, A, B, C)



Use nulls



Movie (<u>title</u>, <u>year</u>, length, weapon, who)

Comparison of approaches answering queries

It is more expensive to answer queries involving several relations

- Nulls approach has advantage

“What movies of 2020 where longer than 150 minutes?”

- In E/R, only the Movie relation is needed
- In object-oriented, all the relations are needed

“What weapons were used in cartoons of over 150 minutes in length?”

- In E/R, the Movie, Cartoon and Crime relations are needed
- In object-oriented, only the MovieCartoonCrime is needed

Comparison of approaches in space

Object-oriented

- Only one tuple per object with components for only the attributes that makes sense
- Minimum possible space usage

Use nulls

- Only one tuple per object but these tuples are “long”, they have components for **all** attributes
- Used space depends on the attributes not being used

E/R approach

- Several tuples for each object but only the key attributes are repeated
- Can use more or less space than the nulls method

Generalizations

E/R style good for

- overlapping generalizations with a large number of subclasses

Object-oriented good for

- disjoint generalizations
- superclass has few attributes and subclasses many attributes

Use nulls good for

- heavily overlapping generalizations with a small number of subclasses

UML key concepts

~~Classes~~

~~Associations~~

~~Association Classes~~

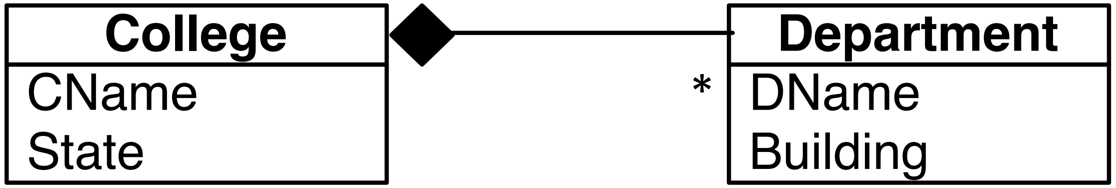
~~Generalizations~~

Composition & Aggregation

Constraints

Derived Elements

Composition

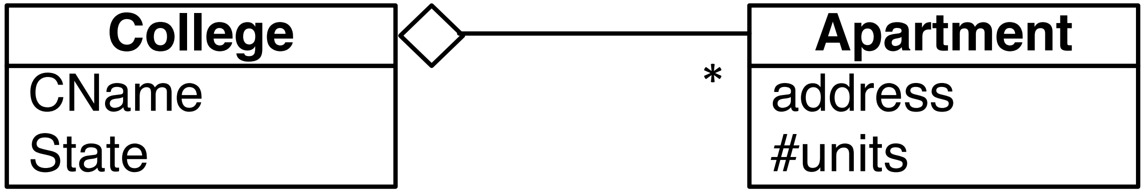


Treat it as a regular association

College (<u>CName</u>, State)

Department (<u>DName</u>, Building, CName->College)

Aggregation



Treat it as a regular association

College (CName, State)

Apartment (address, #units, CName->College)

NULL

UML key concepts

~~Classes~~

~~Associations~~

~~Association Classes~~

~~Generalizations~~

~~Composition & Aggregation~~

Constraints

Derived Elements

Constraints and Derived Elements

Constraints

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
    - Ensures that the value in a column meets a specific condition
- DEFAULT
    - Specifies a default value for a column

Derived Elements

- Treat them as regular elements

Kahoot time!

Any doubts?

Readings

Jeffrey Ullman, Jennifer Widom, A first course in Database Systems 3rd Edition

- Section 2.1 – Basics of the Relational Model
- Section 4.8 – From UML Diagrams to Relations
- Section 4.6 – Converting Subclass Structures to Relations