UML vs E-R

Βάσεις Δεδομένων

Πολυτεχνείο Κρήτης

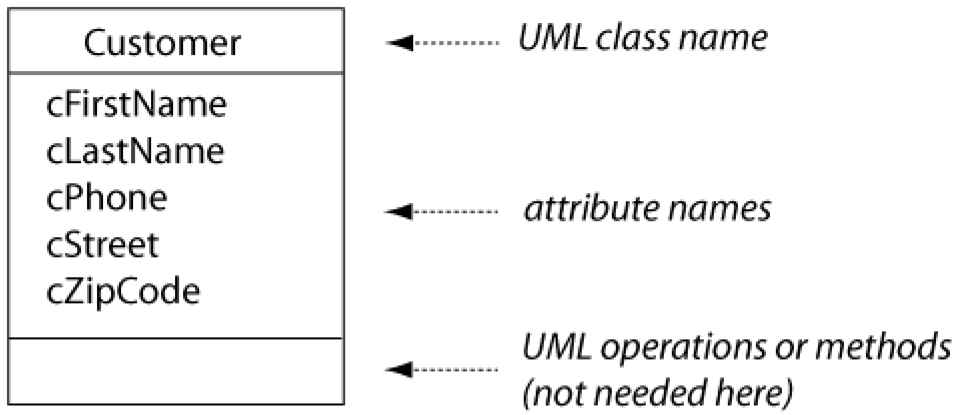
Models and languages

- The **Unified Modeling Language** (UML) was designed for software engineering of large systems using object-oriented (OO) programming languages. UML is a very large language; we will use only a small portion of it here, to model those portions of an enterprise that will be represented in the database. It is our tool for communicating with the client in terms that are used in the enterprise.
- The **Entity-Relationship** (ER) model is used in many database development systems. There are many different graphic standards that can represent the ER model. Some of the most modern of these look very similar to the UML class diagram, but may also include elements of the relational model.

Classes and schemes

- A UML **class** (ER term: **entity**) is any “thing” in the enterprise that is to be represented in our database. It could be a physical “thing” or simply a fact about the enterprise or an event that happens in the real world.
- Each class is uniquely defined by its set of **attributes** (UML and ER), also called *properties* in some OO languages. Each attribute is one piece of information that characterizes each member of this class in the database. Together, they provide the structure for our database tables or code objects.
- In UML, we will only identify **descriptive attributes**—those which actually provide real-world information (relevant to the enterprise) about the class that we are modeling. (These are sometimes called *natural attributes*.) We will *not* add “ID numbers” or similar attributes that we make up to use only inside the database.

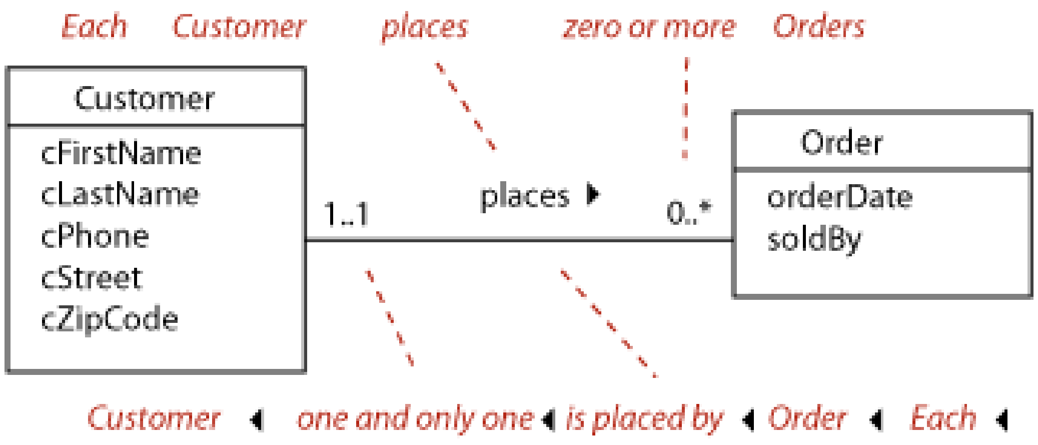
Classes and schemes

- The class diagram shows the class name (always a singular noun) and its list of attributes.
- 

Associations

- The UML **association** (ER term: **relationship**) is the way that two classes are functionally connected to each other.
- Need to include information about how few (at minimum) and how many (at maximum) individuals of one class may be connected to a *single* individual of the other class. This is called the **multiplicity** of the association (ER term: **cardinality**), and we describe it in both directions.

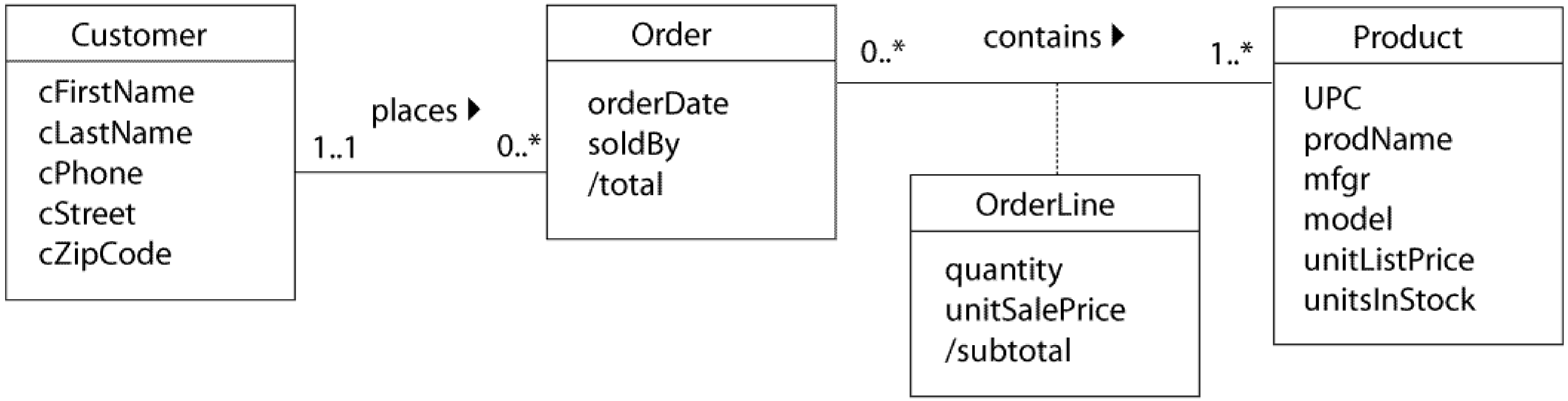
Class diagram



In the diagram, the association is simply shown by a line connecting the two class types. It is named with a verb that describes the action; an arrow shows which way to read the verb. Symbols at each end of the line represent the multiplicity of the association, as we described it above.

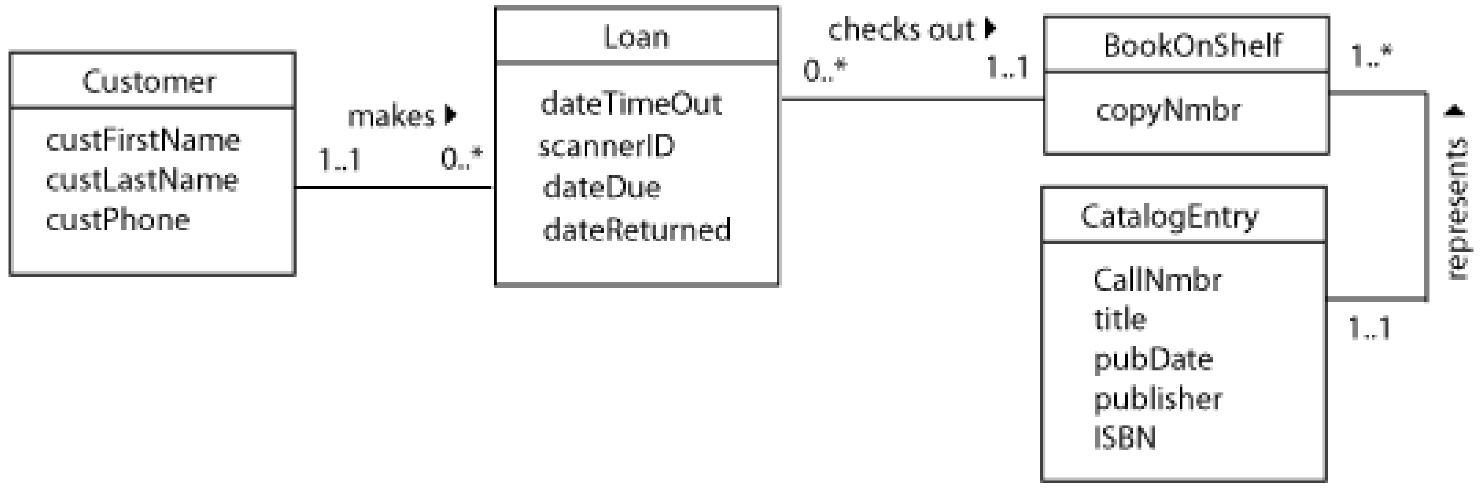
Looking at the *maximum* multiplicity at each end of the line (1 and * here), we call this a **one-to-many** association.

Uml design – many to many



- There are some modeling situations that you will find over and over again as you design real databases. We refer to these as **design patterns**.
- Since the maximum multiplicity in each direction is “many,” this is called a **many-to-many** association between Orders and Products.
- Each time an order is placed for a product, we need to know how many units of that product are being ordered and what price we are actually selling the product for. These attributes are a result of the association between the Order and the Product. We show them in an **association class** that is connected to the association by a dotted line. If there are no attributes that result from a many-to-many association, there is no association class.
- /total, /subtotal: derived attributes

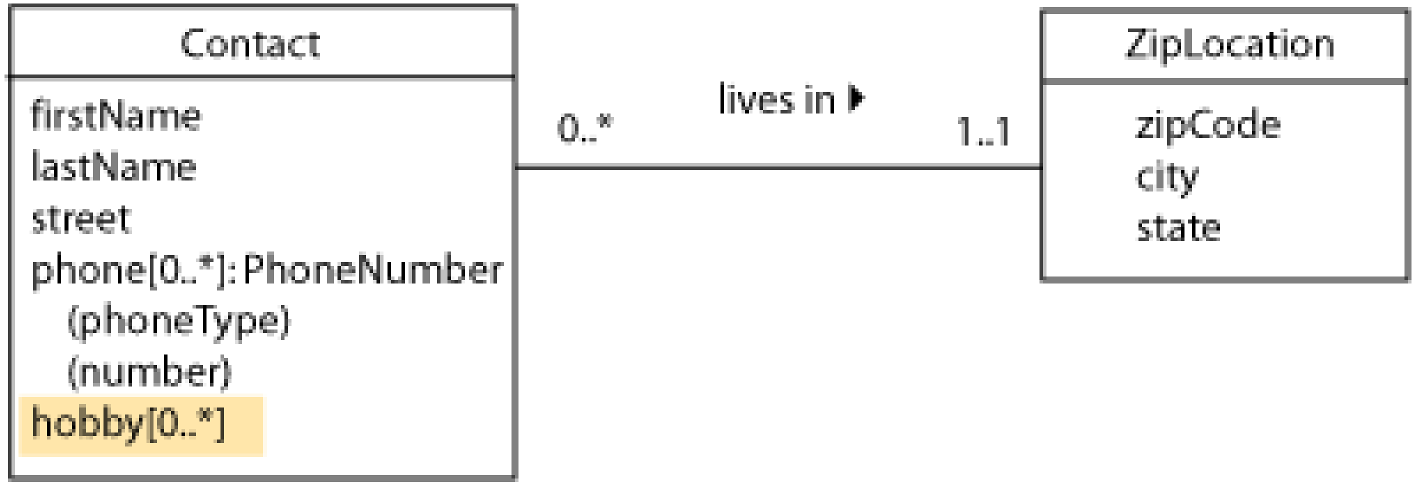
Uml design – many to many



Remember that the UML association class represents the attributes of a many-to-many association, but can only be used if there is at most one pairing of any two individuals in the relationship. This means, for example in order entry, that there can be only one order line for each item ordered. This constraint is consistent with the enterprise being modeled.
There are times when we need to allow the same two individuals in a many-to-many association to be paired more than once. This frequently happens when we need to keep a history of events over time.

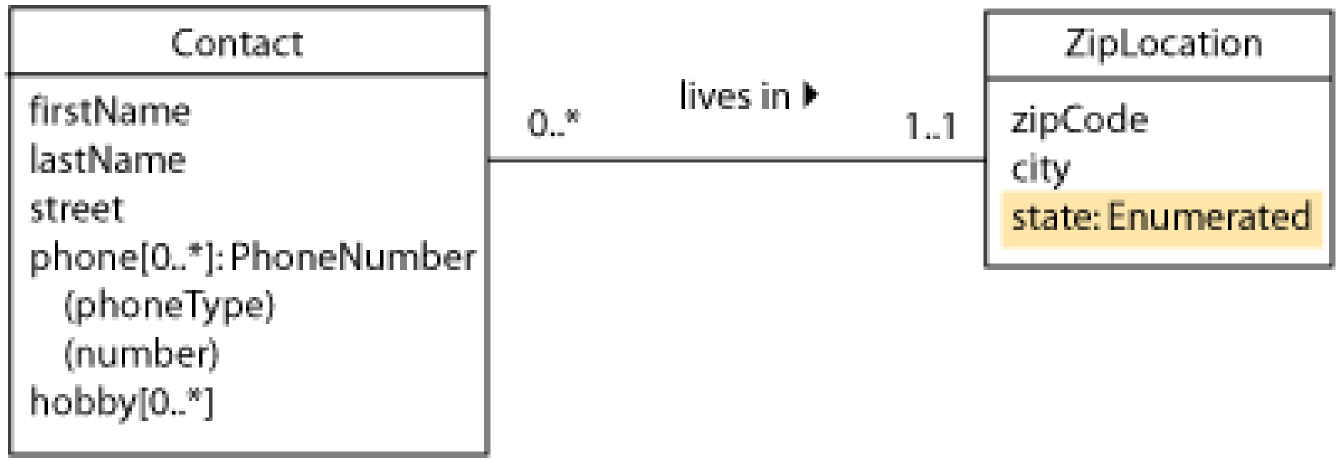
Multivalued attributes

- In UML, we can again use the multiplicity notation to show for example that a contact may have more than one hobby:



Enumerated domains

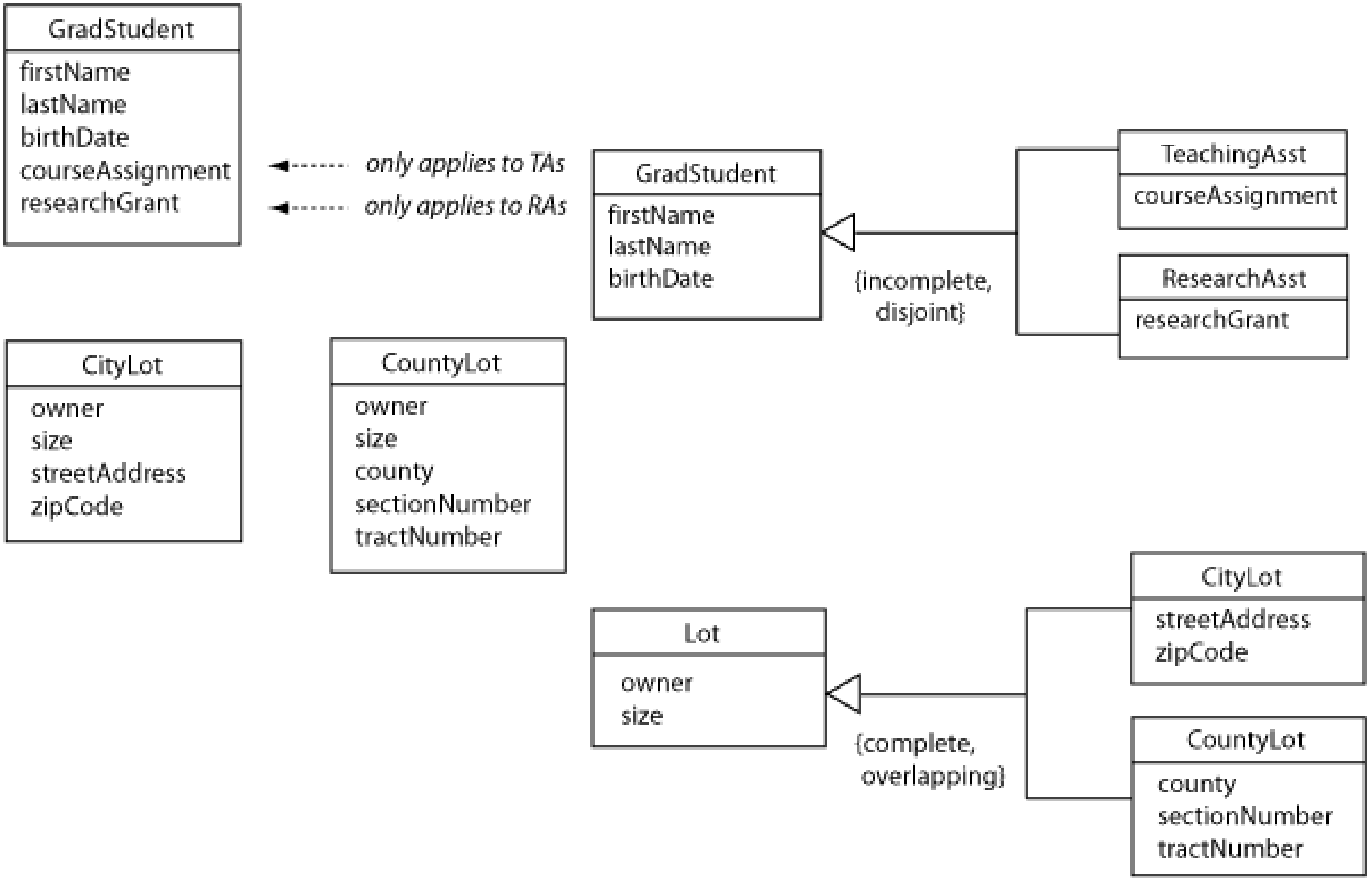
- Attribute domains that may be specified by a well-defined, reasonably-sized set of constant values are called **enumerated domains**. You might know all of the values of the domain at design time, or you might not. In either case, you should keep the entire list of values in a separate table.
- In UML, we can simply use a data type specification to show this, without adding a new class type.



Subclasses

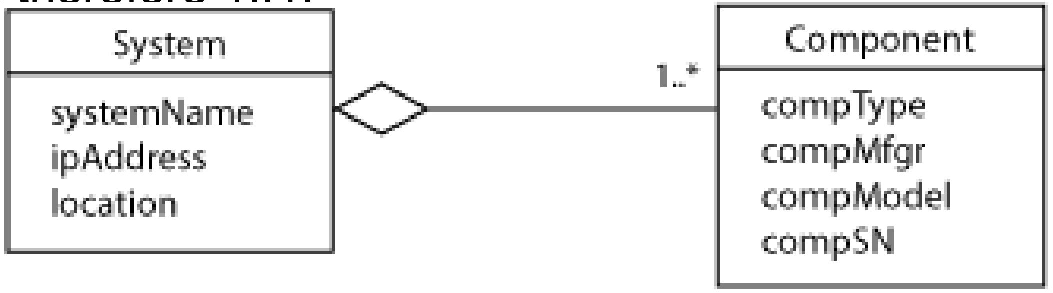
- Top down design: subclass, specialization
- Bottom up design: superclass, generalization
- Constraints
  - In an **incomplete** specialization, also called a **partial** specialization, only *some* individuals of the parent class are specialized (that is, have unique attributes). Other individuals of the parent class have only the common attributes.
  - In a **complete** specialization, *all* individuals of the parent class have one or more unique attributes that are not common to the generalized (parent) class.
  - In a **disjoint** specialization, also called an **exclusive** specialization, an individual of the parent class may be a member of *only one* specialized subclass.
  - In an **overlapping** specialization, an individual of the parent class may be a member of *more than one* of the specialized subclasses.

Subclasses


GradStudent
firstName
lastName
birthDate
courseAssignment
researchGrant
only applies to TAs
only applies to RAs
GradStudent
firstName
lastName
birthDate
{incomplete, disjoint}
TeachingAsst
courseAssignment
ResearchAsst
researchGrant
CityLot
owner
size
streetAddress
zipCode
CountyLot
owner
size
county
sectionNumber
tractNumber
Lot
owner
size
{complete, overlapping}
CityLot
streetAddress
zipCode
CountyLot
county
sectionNumber
tractNumber

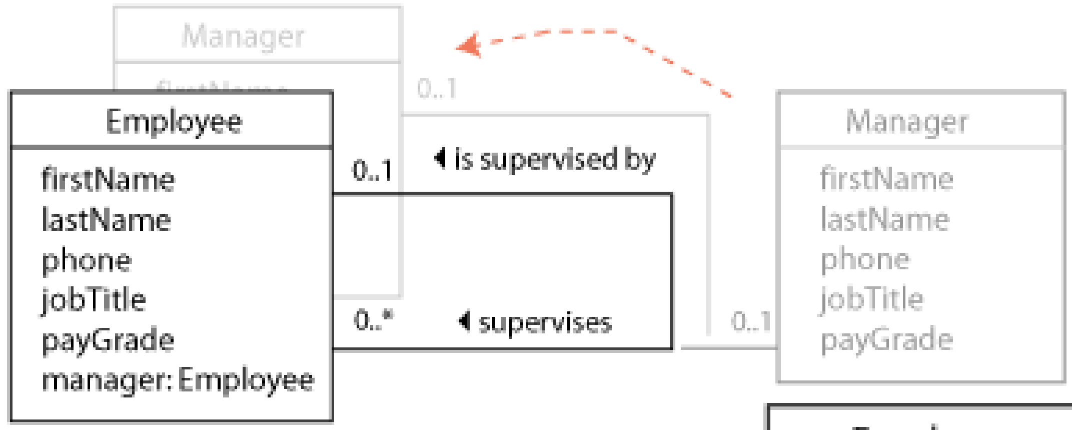
Aggregation

- Sometimes a class type really represents a collection of individual components. Although this pattern can be modeled by an ordinary association, its meaning becomes much clearer if we use the UML notation for an **aggregation**.
- The system is an aggregation of components. In UML, aggregation is shown by an open diamond on the end of the association line that points to the parent (aggregated) class. There is an implied multiplicity on this end of 0..1, with multiplicity of the other end shown in the diagram as usual. To describe this association, we would say that each system is composed of one or more components and each component is part of zero or one system.
- In UML, there is a stronger form of aggregation that is called **composition**. The notation is similar, using a filled-in diamond instead of an open one. In composition, component instances cannot exist on their own without a parent; they are created with (or after) the parent and they are deleted if the parent is deleted. The implied multiplicity on the "diamond" end of the association is therefore 1..1.

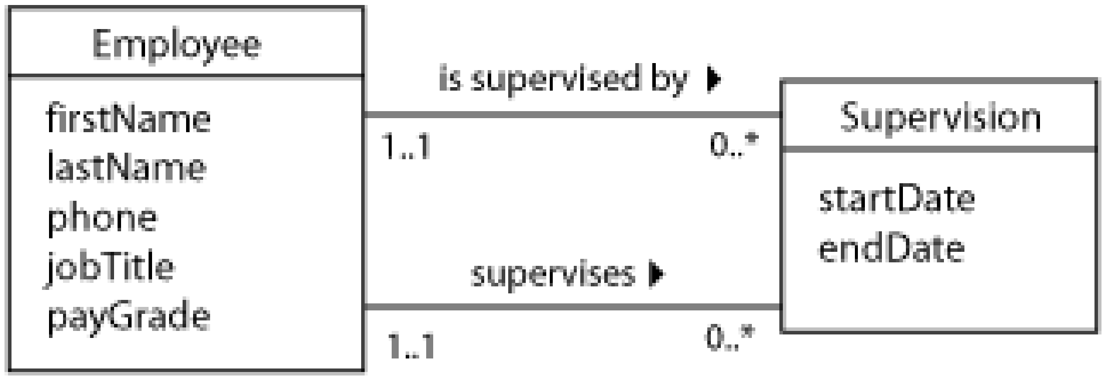


Recursive associations

- A **recursive association** connects a single class type (serving in one role) to itself (serving in another role).



In some project-oriented companies, an employee might work for more than one manager at a time. We also might want to keep a history of the employees' supervision assignments over time.



Παράδειγμα εφαρμογής ΒΔ

Εφαρμογή ΒΔ για την καταγραφή των υπαλλήλων, των τμημάτων και των έργων μιας εταιρείας με τις ακόλουθες απαιτήσεις δεδομένων:

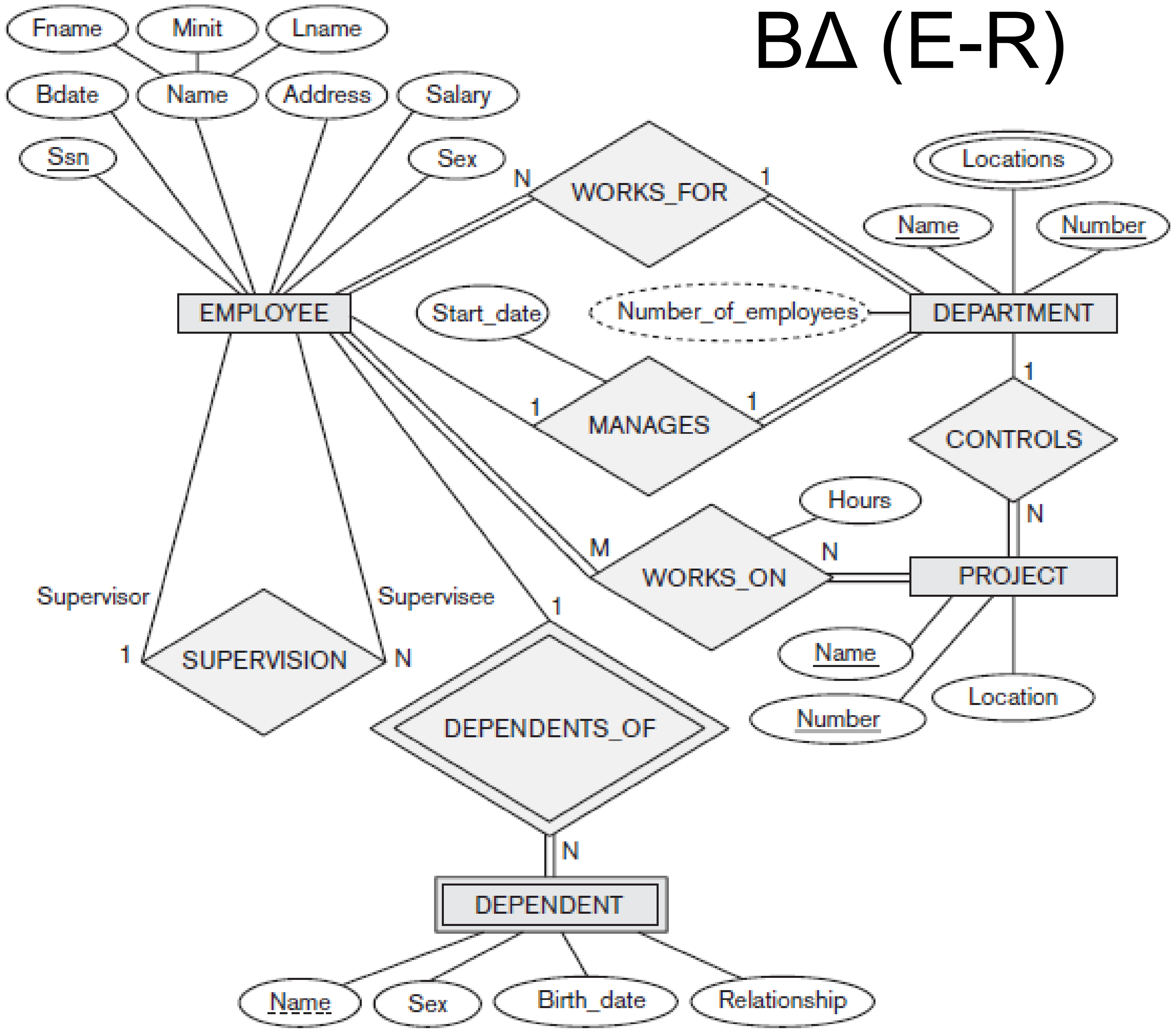
- Η εταιρεία είναι οργανωμένη σε τμήματα. Κάθε τμήμα έχει ένα μοναδικό όνομα, ένα μοναδικό αριθμό και ένα συγκεκριμένο υπάλληλο που το διευθύνει. Καταγράφεται η αρχική ημερομηνία που ο υπάλληλος αυτός ανέλαβε τη διεύθυνση του τμήματος. Το τμήμα μπορεί να βρίσκεται σε πολλαπλές τοποθεσίες.

- Ένα τμήμα ελέγχει ένα αριθμό από έργα καθένα από τα οποία έχει ένα μοναδικό όνομα, ένα μοναδικό αριθμό και εκτελείται σε μία μοναδική τοποθεσία.

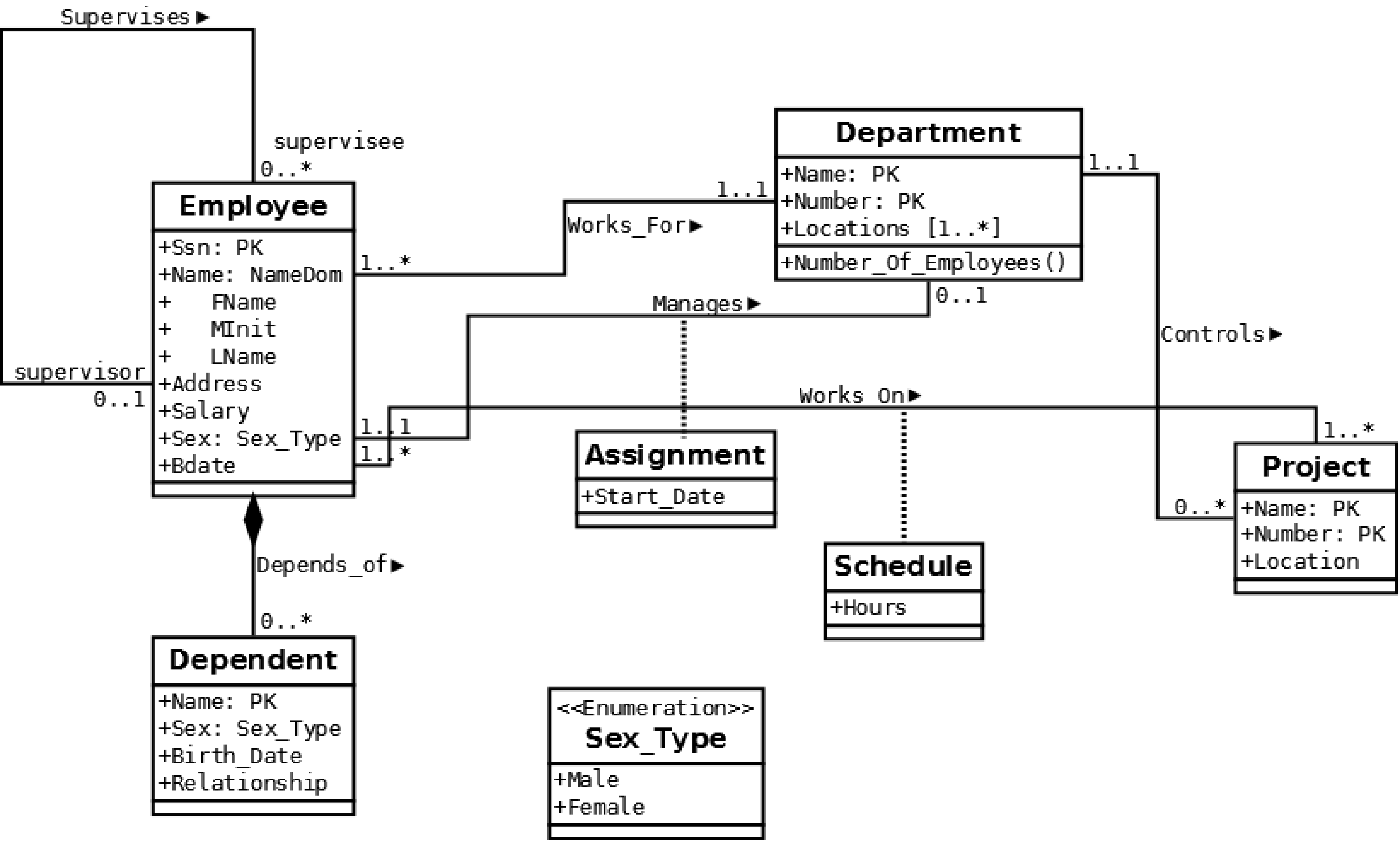
- Καταγράφεται για κάθε υπάλληλο το πλήρες όνομά του, ο ΑΜΚΑ του, η διευθύνση του, ο μισθός, το φύλο και η ημερομηνία γέννησης. Ένας υπάλληλος ανήκει σε ένα τμήμα αλλά μπορεί να απασχολείται σε πολλά έργα τα οποία δεν ελέγχονται απαραίτητα από το ίδιο τμήμα. Καταγράφεται ο τρέχων αριθμός ωρών ανά εβδομάδα που ένας υπάλληλος απασχολείται σε κάθε έργο. Επίσης καταγράφεται ο προϊστάμενος κάθε υπαλλήλου (που θεωρείται επίσης υπάλληλος).

- Καταγράφονται τέλος τα εξαρτώμενα μέλη κάθε υπαλλήλου και συγκεκριμένα για κάθε μέλος κρατείται πληροφορία για το όνομα, το φύλο, την ημερομηνία γέννησης και τη συγγένεια του με τον υπάλληλο.

Παράδειγμα εφαρμογής ΒΔ (E-R)



Παράδειγμα εφαρμογής ΒΔ (UML)


Supervises ►
supervisee
0..*
supervisor
0..1
Employee
+Ssn: PK
+Name: NameDom
+ FName
+ MInit
+ LName
+Address
+Salary
+Sex: Sex_Type
+Bdate
1..*
Works_For ►
1..1
Department
+Name: PK
+Number: PK
+Locations [1..*]
+Number_Of_Employees()
1..1
0..1
Manages ►
Controls ►
1..1
1..*
Works On ►
1..*
Assignment
+Start_Date
Project
+Name: PK
+Number: PK
+Location
0..*
Schedule
+Hours
Depends_of ►
0..*
Dependent
+Name: PK
+Sex: Sex_Type
+Birth_Date
+Relationship
<<Enumeration>>
Sex_Type
+Male
+Female