ER Diagram to UML Class Diagram

Motivation: An indirect approach to find classes from a software-engineering problem is through the construction of ER diagrams. Constructing an ER diagram ensures that the sets of elements with significant properties are selected. As a next step, we convert each entity type into a class, where each class represents sets of elements with significant properties. However, a class is supposed to indicate only those sets of elements (with significant properties) that offer different services to the whole software. Hence, we first convert the ER diagram to a *crude* class diagram. Then, using the information from other sources like use case diagrams and sequence diagrams, we identify the services offered by each class. In this process if we find that more classes are needed to offer all essential services, we create new classes. In contrast, if we find that there are classes that are offering no services or redundant services, we try to merge these classes with other existing classes. In other words, we eventually remove those classes.

Overview: The class diagrams of UML (Unified Modeling Language) are similar to ER diagrams in many ways. Unfortunately, the terminology often differs. An *entity type* in an ER diagram corresponds to a *class* in the class diagram. Although an *entity* in an ER diagram loosely corresponds to an *object* in UML, the class diagram *never* shows any *object*. In the following part, we discuss how the symbols/notations of ER diagrams are mapped into the corresponding items of the class diagrams.

ER to UML class diagram conversion:

1. **Enity-type**

- *Strong entity type:* A strong entity type becomes a class, which is displayed as a box. A class includes three sections: the top section gives the class name; the middle section includes the attributes for individual objects of the class; and the last section includes "operations" that can be applied to these objects. We will discuss the "operations" later, and hence for the time being we keep it blank. The class diagram shows the primary key using a notation <<PK: …>> . The notation <<..>> , which is also know as a *stereotype*, is important. Using a stereotype, we can show many different concepts for which we do not have any proper notation.
  Example: Employee in Fig. 3 and Fig. 4.

- *Weak entity type:* It also becomes a class. The partial key of the weak entity type is shown in a projected box. This projected box is *not* attached to the class obtained from the weak entity type. Rather the projected box appears in the class that corresponds to the strong entity type on which the weak entity type depends. In a class diagram, the weak relationship is called *qualified association*.
  Example: Dependent in Fig. 3 and Fig. 4.

2. **Attributes:**

- *Simple:* They are shown in the class box.
  Example: Attributes of Employee in Fig. 3 and Fig. 4.

- *Composite:* Show all the components serially, with their combined name at the beginning.
  Example: Names of the class Employee in Fig. 3 and Fig. 4.

- *Derived:* Just use '/' in front of the attribute name.
  Example: noOfEmployee of Department in Fig. 3 and Fig. 4.

- *Multivalued:* Here we use a concept called *aggregation*. We form a separate class (say class A) for each multivalued attribute, and connect this class with the original class using aggregation symbol (a diamond). Note that the aggregation symbol is attached to the class corresponding to the original entity type, and it is not the other way. However, there are two variations: the diamond could be either dark or hollow. If we pick one element from the class A and we find that the element can be related to *only one* element from the original class, and the existence of A depends on the original class, then it is called *composite aggregation* or *composition*. It is shown using a dark diamond. One example is our fingers: A given finger is a part of a *particular* human body, and without the body there is just no existence of any finger. Let us see the other case now. If we pick one element from the class A, and we find that the element can be related to *more than one* element from the original class, then it is called *shared aggregation*. It is shown using a hollow diamond. One example is the connection between location and department in Fig. 3 and Fig. 4. Each department should have minimum one location, and it can have more than one location. Each location can be occupied by minimum one department, and the same location can be shared by more than one department (Fig. 3 and Fig. 4). Hence, it is a shared aggregation. Note that here even if the department does not exist, the location exists. In other words, we decide the type of the aggregation depending on the cardinality and participation constraint (i.e., one-to-one, one-to-many, many-to-one, many-to-many, total, partial). We also show the cardinality and participation constraint in the class diagram.

3. Relations:

   Binary relations: The relationships in UML terminology are called *associations*. In the class diagram, the cardinality is referred to as *multiplicity*.

   - *One-to-one, many-to-one, one-to-many relations that are either total or partial:* In a class diagram all relations are denoted by using a straight line. The name of the relation is written above this line. You may also prefer to write the *role* names. Each side of the association contains some interval that shows both the multiplicity and participation constraint.

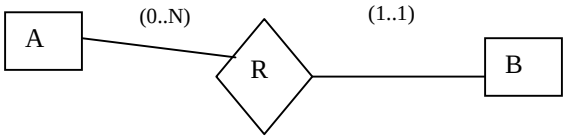

Fig. 1: A part of an ER diagram.

The cardinality and participation constraint of an association is shown using an interval. Here, the difference between ER diagram and the class diagram are (a) instead of writing 1..N, we write 1..*, and (b) the positioning of this interval is just opposite to that of in ER diagram. The class diagram corresponding to Fig. 1 is shown in **Fig. 2**.

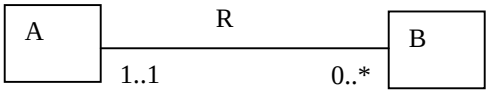

Fig. 2: Class diagram corresponding to Fig. 1.

Example: Works_for association in Fig. 3 and Fig. 4.
Some books would like to use one symbol(►) after works_for. It is optional. It is used to show the meaning that employees work for departments; but departments do not work for employees.

- *Many-to-many relations that are either total or partial:* Make a separate class for it, and show all attached attributes inside the class. Remember that this class contains the primary key of both the connecting classes. The technique to show multiplicity and participation coefficient is same as above.
  Example: The association Manages in Fig. 3 and Fig. 4.

- *Descriptive attribute:* If the relationship, which is either one-to-one or many-to-one or one-to-many, has some descriptive attribute, first make a separate class for it, and show all attached attributes inside the class. The descriptive attributes (i.e., the attributes connected to the relations) are later either exported to the entity type at the left-hand side or exported to the entity type at the right-hand side. Exact entity type depends on the relationship. We decide it following the techniques that we adopt while converting a relationship into a table. From those techniques, we know that we do not construct any table for one-to-one, one-to-many, and many-to-one relationships. We export the descriptive attributes to the "many" sides of the one-to-many or many-to-one relationships. If the relationship is one-to-one, and it has total relationship only at one side, we export the descriptive attributes to the class that is connected through the total relationship. If both or none of the sides of the one-to-one relationship are total, then all the descriptive attributes are exported to the entity type either at the left hand side or at the right hand side.

If the relationship is many-to-many, then we always create a class. Hence, if the relationship has some descriptive attribute, we show it in the class box.
Example is Works_for in Fig. 3 and Fig. 4.

- *Recursive relation:* In the class diagram, the recursive relation is called *reflexive association*. Although writing the role name is not that important in case of ordinary associations, it is very important in case of reflexive associations. Otherwise, we do not understand which part of the association means what.
  Example: The reflexive association supervision in Fig. 3 and Fig. 4 has two roles supervisor and supervisee.

- *N-ary relations (N>2):* It is similar to the ER diagram. Only difference is that the multiplicity constrains are shown using UML notations.

6. *Aggregations:* There is no direct way to represent the aggregations in UML diagram. Remember how you convert an aggregation into a table. Do it. Then represent each table using a class. Note that this aggregation, i.e., the term aggregation defined in the context of an ER diagram, is different from the aggregation term used in the class diagram.

7. *Generalization and specialization:* Create a class for each super and sub types. Connect the subtypes to the supertype using an arrowhead. If the generalization is
total, use {mandatory}
partial, use {optional}
disjoint use {and}
overlapping use {or}

The subclasses inherit all the attributes of the superclass. Moreover, if the generalization is total, then the superclass is called *abstract class.* The name of the abstract class is shown using italics.

Example: the entity types research and industrial projects are generalized using the entity type projects. Hence, we also got two classes research and industrial projects. Since the generalization is total and disjoint, we attach a comment {mandatory, and}. In addition, project is an abstract class, and hence it is in Italics.

Note that neither the ER diagram nor the class diagram is unique. The UML class diagram corresponding to Fig. 3 is shown Fig. 5. This diagram shows how the UML class diagram looks like just after the conversion; however, it does not show the refined class diagram. We can obtain the refined class diagram only after consulting use case diagram and sequence diagram.

Can we delete some of the classes that appeared due to the descriptive attributes of the one-to-one, one-to-many and many-to-many relations? Using the same technique that we follow while converting an ER-diagram to a set of tables, we get the modified class diagram as shown in Fig. 5. Note that the class corresponding to Manages has been merged with the class Department.

Table 1: The notational differences between ER diagrams and class diagrams are shown.

Item	ER Diagram	Class Diagram
Strong entity type or class	Box	Box
Weak entity type or reflexive class	Two concentric boxes	Box.
Simple attribute or variables	Bubble attached to the entity type	Text inside class box
Composite attribute or variable	Bubbles attached to a main bubble	Structured text inside class box
Derived attribute or variable	Bubble with dotted line	'/' before the attribute name
Multivalued attribute or variable	Two concentric bubbles	Open or filled diamond, i.e., aggregation/composition
Primary key	Underlined attribute name	Use stereotype <<PK>>
Partial key	Dotted underline	One projected box is attached with the class on which the reflexive class depends. This box contains the partial key.
Recursive relationship or reflexive association	Self join through a diamond	Self join using straight lines
Binary relationship or association	Diamond	Straight line
N-ary (N>2) relationship	Diamond	Diamond
Relationship with descriptive attributes or variables	Bubbles coming out from the relationship	Class attached to association with a dashed line. This class contains the variables.
Cardinality or multiplicity	Interval.	Interval, but the positions of the intervals are just opposite to that in ER diagram. Instead of writing interval (1..N), here we write 1..* or (1..*).
Generalization/specialization	Circle	Arrow with hollow head
Disjoint generalization	'D' inside the circle	Add comment {And}
Overlapping generalization	'O' inside the circle	Add comment {Or}
Total generalization	Two parallel lines	Add comment {Mandatory}
Partial generalization	One line	Add comment {Optional}

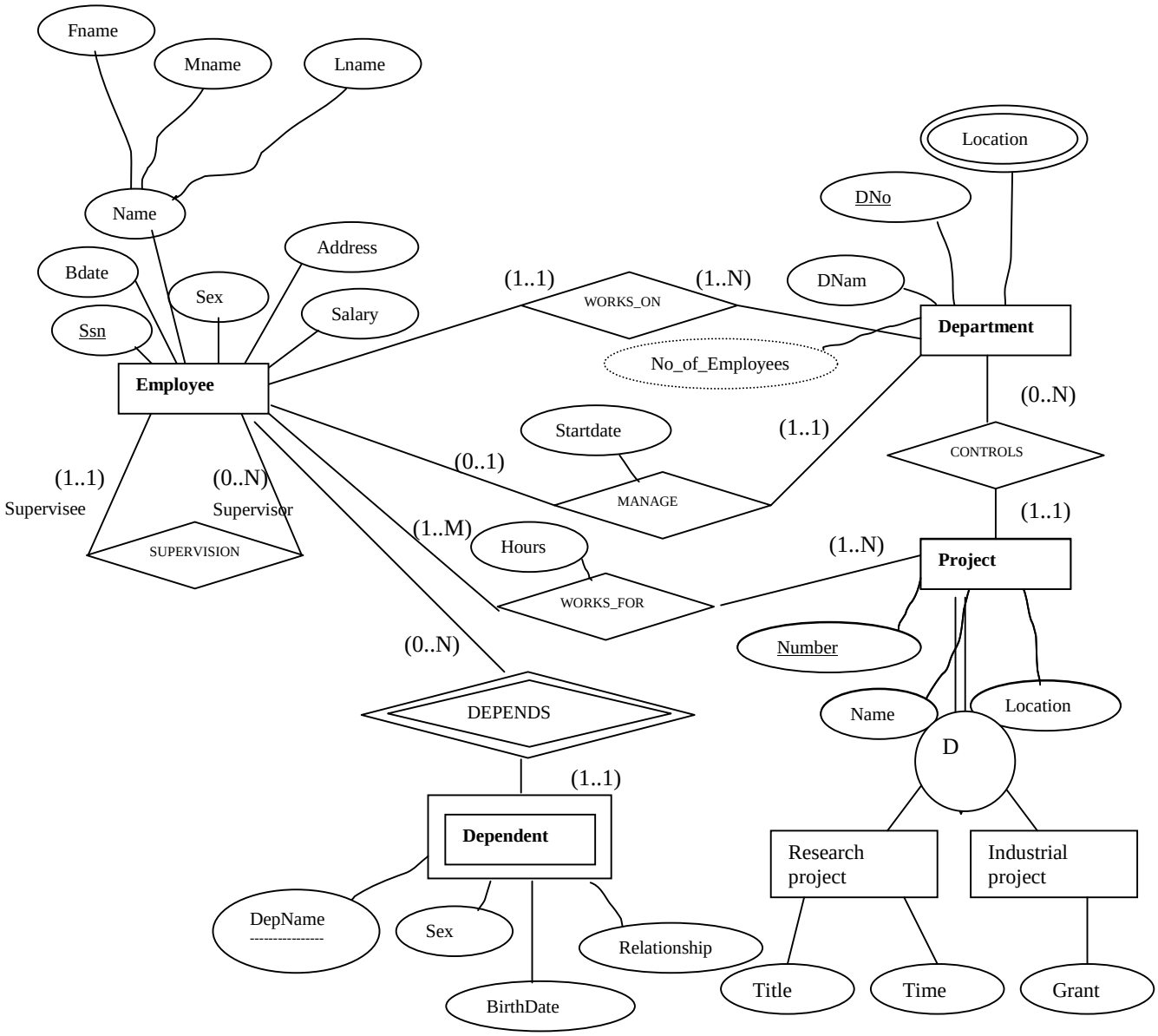

Fig. 3: An ER diagram for which we want the corresponding class diagram.

Fig. 4: A possible class diagram for **Fig. 3**. Note that neither the ER diagram nor the class diagram is unique. The corresponding UML class diagram is shown below. This diagram shows how the UML class diagram looks like just after the conversion; however, it does not show the refined class diagram. We can obtain the refined class diagram only after consulting use case diagram and sequence diagram.

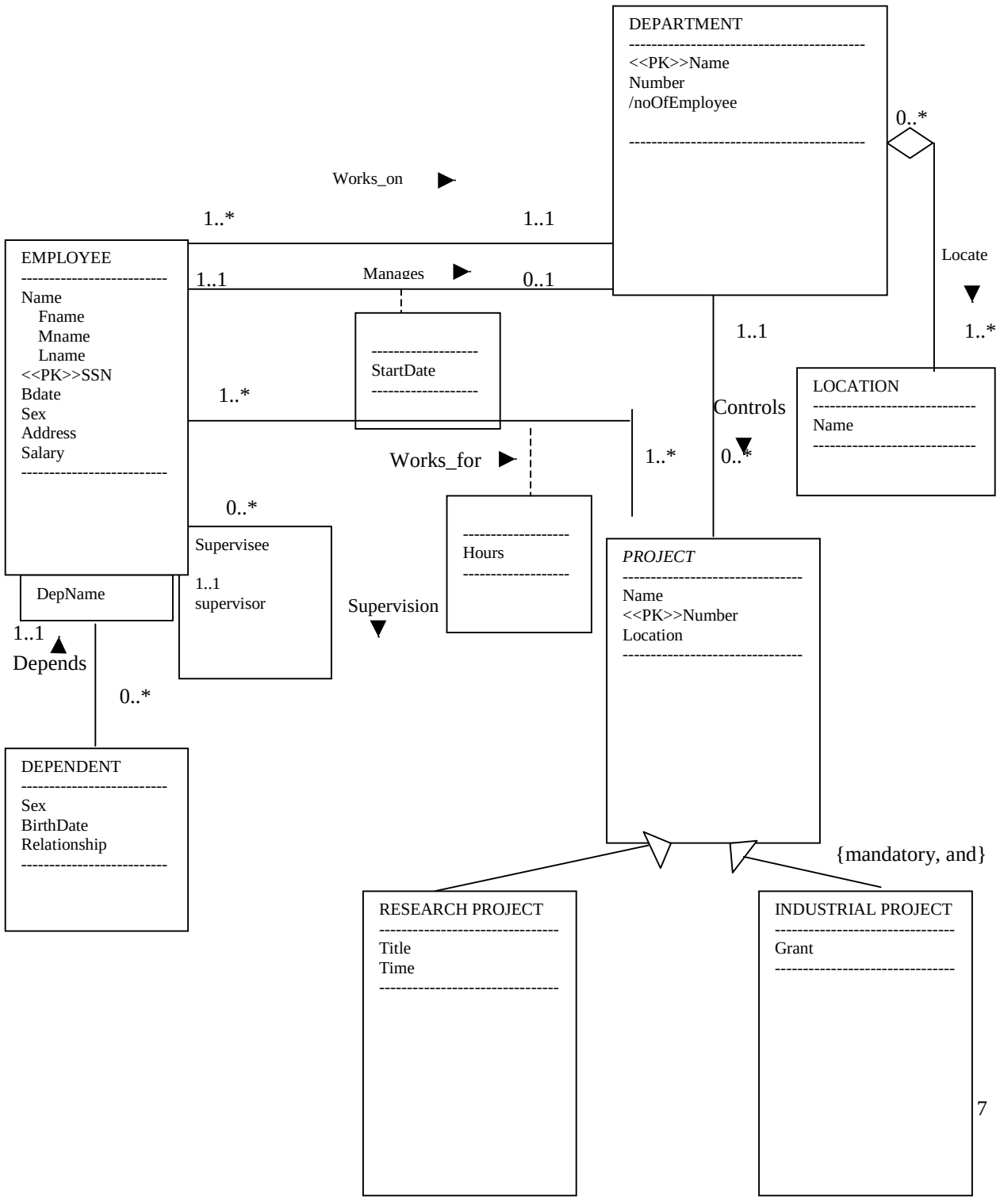

Fig. 5: A modified class diagram for **Fig. 4:**, where the classes corresponding to one-to-one-one to-many, and many-to-many relationships have been merged with other classes. In this figure, the class corresponding to Manages has been merged with the class Department.