Unified Modeling Language (UML)

APPENDIX H

Preview

The Unified Modeling Language (UML) is an object-oriented modeling language sponsored by the Object Management Group (OMG) and published as a standard in 1997. UML is the result of an effort headed by the OMG to develop a common set of object-oriented diagrams and notations (symbols and constructs) for the analysis, design, and modeling of systems. Because the origin of UML is closely related to the object-oriented concepts you explored in Appendix G, Object-Oriented Databases, object terminology is used throughout this section.

Keep in mind that UML is not a methodology or procedure for developing databases. Rather, UML is a language that describes a set of diagrams and symbols that can be used to model a system graphically. UML diagrams encompass static data components (classes and their associations) and components such as business processes, data flows, and hardware. Table H.1 shows the nine different types of diagrams that the UML standard offers.

TABLE H.1 UML Diagrams

DIAGRAM NAME	USAGE
Activity diagram	Describes the behavior of a system. Very similar to data flow diagrams that model specific business processes. Related to Use Case diagrams.
Class diagram	Describes the static components of object classes. (Remember, a class is a collection of similar objects.) Similar in function to an ER diagram in a relational database modeling.
Collaboration diagram	Describes the interaction between objects in a system—messages sent among objects, parameters passed, actions taken, and so on. An alternative to the Sequence diagram.
Component diagram	Describes the arrangement of software components that form a system and the way those components interact.
Deployment diagram	Describes the arrangement of hardware components within a system. Describes what objects run in each component.
Sequence diagram	Describes the interaction between objects in a system—(what messages are invoked and in what order). Very similar to Collaboration diagrams and related to Use Case diagrams.
State diagram	Describes the object's state during object interactions. Models the changes in an object's state during its interactions with other objects.
Object diagram	Describes the static nature of object instances within a system at a given point in time.
Use Case diagram	Describes business processes within a system. Very similar to data flow diagrams.

Because the main focus here is on database design, not all of the different types of diagrams that UML offers are covered. Instead, the content focuses on the use of Class diagrams to model the static data components (object classes and their relationships) that are part of a database *system*.

H.1 Using Class Diagrams to Model Database Tables

The UML Class diagram is the equivalent of the ER diagram in the relational model. The Class diagram is used to model object classes and their associations. Because an object class is a collection of similar objects, a class is the equivalent of an entity set in the ER model. Therefore, a class is described by its attributes—and by its methods.

NOTE

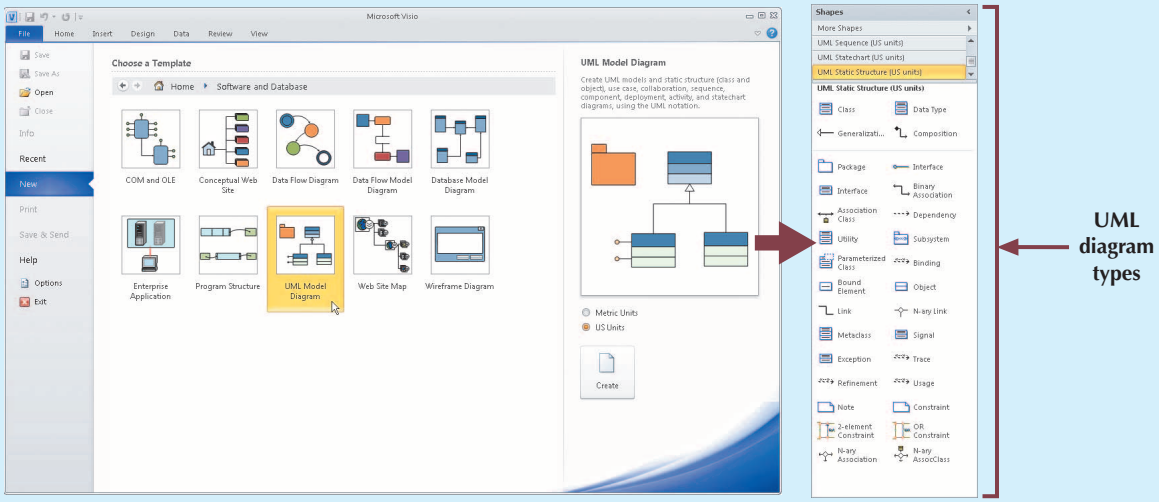
MS Visio Professional has been used to develop the examples shown in this appendix. To create Class diagrams in MS Visio Professional, from the menu select **File, New, Software, UML Model Diagram, UML Static Structure.** The sequence is illustrated in Figure H.1.

H.1.1 Classes to Represent Entity Sets

In a UML Class diagram, a class is represented by a box that is subdivided into three parts.

1. The top part is used to name the class.
2. The middle part is used to name and describe the class attributes. (A class attribute is identified by a name and a data type.)
3. The bottom part is used to list the class methods. Both the attributes and the methods are displayed.

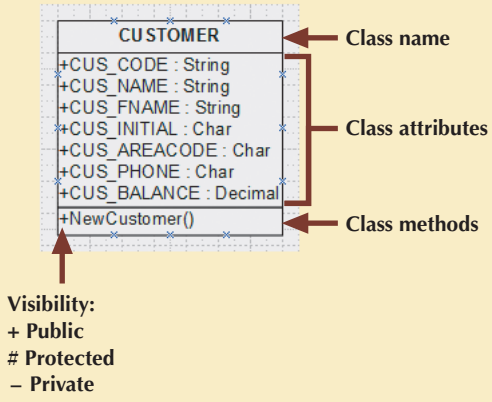
FIGURE H.1 Creating Class diagrams in Visio: Starting the process



SOURCE: Course Technology/Cengage Learning

The three parts are illustrated in Figure H.2.

FIGURE H.2 UML representation of the Customer class



SOURCE: Course Technology/Cengage Learning

As you can see in Figure H.2, the UML representation of a class is very similar to the ER representation of an entity, but there are some important differences.

- A class box also lists the methods of the class in the bottom part of the box.
- A + symbol is placed before attributes and methods. The + symbol indicates the visibility of the UML element.

H.1.2 Visibility

The visibility concept is derived from object-oriented programming. *Visibility* describes the availability of an object attribute or method to other objects or methods. Visibility characteristics are summarized in Table H.2.

TABLE H.2 Attribute and Method Visibility

	PUBLIC (+)	PROTECTED (#)	PRIVATE (-)
Attribute	The attribute is available for read/write purposes to any method of any class.	The attribute is available for read/write purposes *only* to the methods of the class and its subclasses.	The attribute is available only to the methods of the class.
Method	The method can be invoked by any method of any class.	The method can be invoked only by the methods of the class and its subclasses.	The method is available only to the methods of the class.

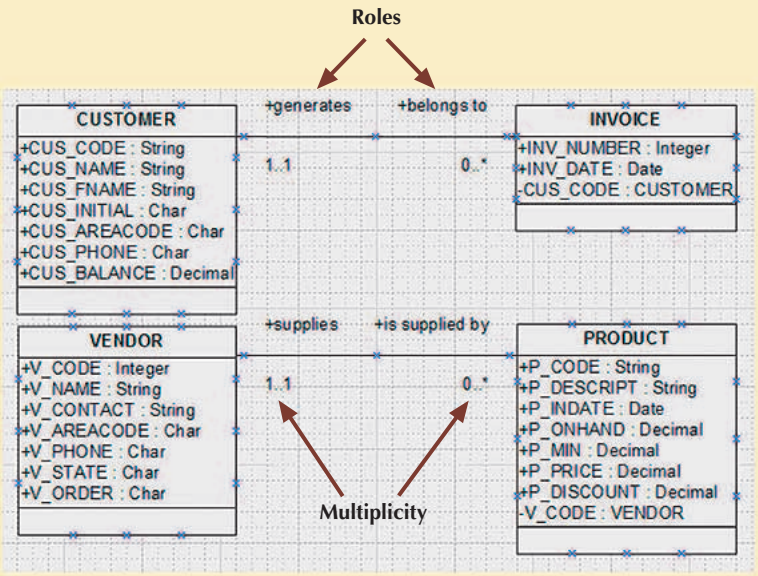
H.2 Associations to Represent Relationships

The UML Class diagram represents relationships as associations among objects. (An object is an instance of a class.) Because associations among classes are critical for database design purposes, you begin by studying how the UML Class diagram represents 1:M associations.

H.2.1 Representing 1:M Associations

Figure H.3 shows a UML Class diagram with two 1:M relationships: a CUSTOMER generates many INVOICEs, and a VENDOR provides many PRODUCTs.

FIGURE H.3 Representing 1:M relationships with Class diagrams



SOURCE: Course Technology/Cengage Learning

NOTE

UML Class diagrams do not require the foreign key attribute to be added to the "many" side of the 1:M relationship. The object-oriented model implements class associations through the use of object IDs, which are internally managed by the OODBMS. (See Appendix G.) However, because the focus here is on the use of UML Class diagrams to model relational databases, the foreign key attributes are shown in the class diagrams.

By examining Figure H.3, you can see that associations are represented by lines that connect the classes. Associations have several characteristics.

- *Association name*. Each association has a name. Normally, the name of the association is written over the association line. In the example, the association name is not shown; instead, role names are used.
- *Role name*. The participating classes in the relationship can also have role names. A role name expresses the role played by a given class in the relationship. In Figure H.3, the role names represent the relationship "as seen" by each class; for example:

  A CUSTOMER *generates* an INVOICE, and each INVOICE *belongs to* a CUSTOMER.

  A VENDOR *supplies* a PRODUCT, and each PRODUCT *is supplied by a* VENDOR.
- *Association direction*. Associations also have a direction, represented by an arrow (→) pointing to the direction in which the relationship flows. (Relationship direction is not displayed in Figure H.3.)
- *Multiplicity*. Multiplicity refers to the number of instances of one class that are associated with one instance of a related class. Multiplicity in the UML model provides the same information as the connectivity, cardinality, and relationship participation constructs in the ER model. For example:
  - One (and only one) CUSTOMER generates zero to many INVOICEs, and one INVOICE belongs to one and only one CUSTOMER.
  - One (and only one) VENDOR supplies zero to many PRODUCTs, and one PRODUCT is supplied by one and only one VENDOR.

Table H.3 shows the different multiplicity values that can be used.

TABLE H.3 Multiplicity

MULTIPLICITY	DESCRIPTION
0..1	A minimum of zero and a maximum of one instance of this class are associated with an instance of the other related class (indicates an optional class).
0..*	A minimum of zero and a maximum of many instances of this class are associated with an instance of the other related class (indicates an optional class).
1..1	A minimum of one and a maximum of one instance of this class are associated with an instance of the other related class (indicates a mandatory class).
1..*	A minimum of one and a maximum of many instances of this class are associated with an instance of the other related class (indicates a mandatory class).
1	Exactly one instance of this class is associated with an instance of the other related class (indicates a mandatory class).
*	Many instances of this class are associated with an instance of the other related class.

The multiplicity symbols implicitly describe the relationship participation concept used in the ER model. For example, a "1..1" multiplicity on the CUSTOMER side indicates a mandatory participation. A "0..*" multiplicity on the INVOICE side indicates an optional participation.

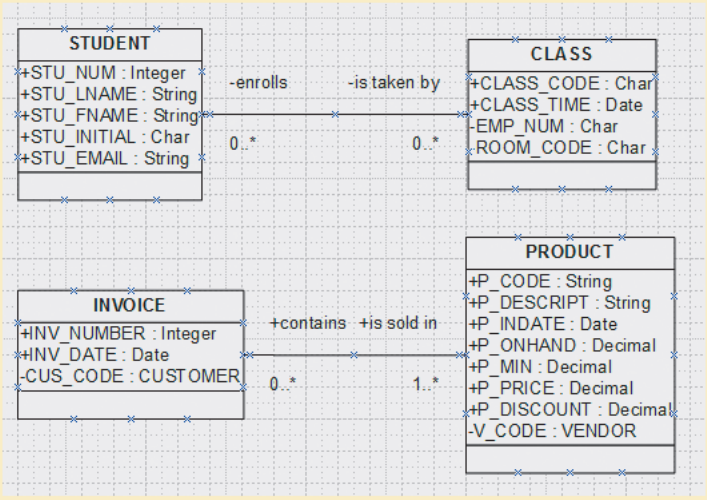
If you examine Figure H.3, you will note that the visibility of the foreign key attributes CUS_CODE and V_CODE have been defined as private (-). Although there is no requirement to specify the foreign key attributes in the UML class diagram, this option has been chosen to highlight the use of the visibility property within the attributes of a class.

H.2.2 Representing M:N Associations

UML Class diagrams can use the multiplicity element to represent M:N relationships directly. For example, Figure H.4 shows the following two examples of M:N associations:

- A STUDENT enrolls in many CLASSes, and each CLASS is taken by many STUDENTs.
- An INVOICE contains many PRODUCTs, and each PRODUCT is sold in many INVOICEs.

FIGURE H.4 M:N associations in a UML Class diagram



SOURCE: Course Technology/Cengage Learning

If you examine Figure H.4, you will see that the M:N association between STUDENT and CLASS is optional at both ends. (The optionality is represented by the "0..*" multiplicity.) However, the M:N association between INVOICE and PRODUCT exhibits two different multiplicity values. The "1..*" multiplicity at the PRODUCT end indicates that an INVOICE contains a minimum of one and a maximum of many PRODUCT instances. The "0..*" multiplicity at the INVOICE end indicates that a PRODUCT is sold in a minimum of zero and a maximum of many INVOICE instances.

In the UML diagram, the multiplicity value always refers to the class to which that value is attached. That is, you always try to find out how many instances of a class are associated to one instance of another class. Contrast that to the use of role names, which are always close to the class that plays the role.

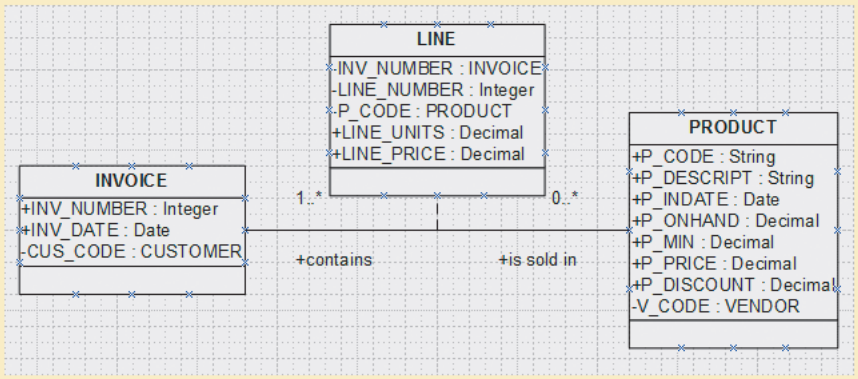
H.2.3 Association Class

An association class is used to represent a M:N association between two classes. The association class exists within the context of the associated objects, and as in the ER model, the association class can have its own attributes. Figure H.5 shows the use of a LINE association class to represent the M:N relationship between INVOICE and PRODUCT.

H.2.4 Composition and Aggregation

The UML Class diagram uses symbols to indicate the strength of the association between two class instances. In particular, the UML Class diagram uses aggregation and composition to indicate the strength of dependency between two classes participating in an association. Table H.4 summarizes the main characteristics of the aggregation and composition UML constructs.

FIGURE H.5 Association class representation



SOURCE: Course Technology/Cengage Learning

TABLE H.4 Aggregations and Compositions

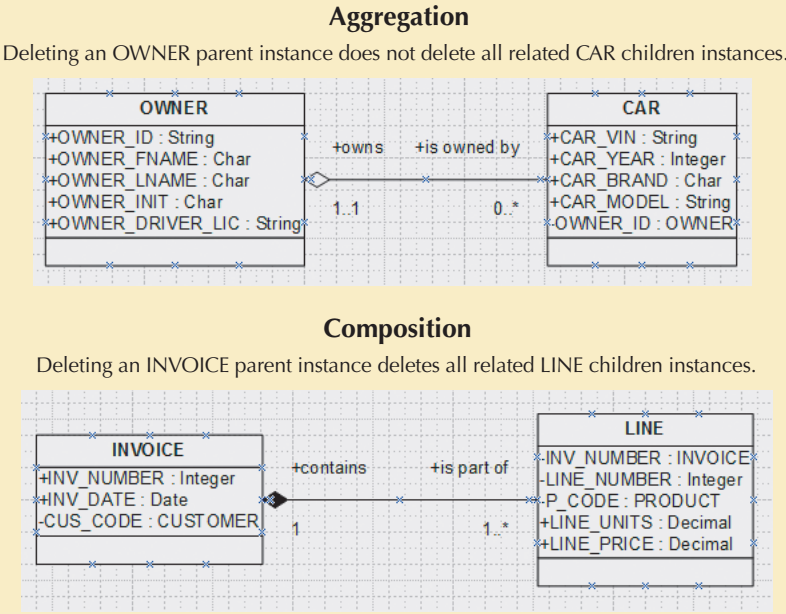
UML CONSTRUCT	UML SYMBOL	DESCRIPTION
Aggregation	◊——	This type of association represents a "has a" type of relationship (that is, an object that is formed as a collection of other objects). An aggregation indicates that the dependent (child) object instance has an optional association with the strong (parent) object instance. When the parent object instance is deleted, the child object instances are not deleted. The aggregation association is represented by an empty diamond in the side of the parent entity.
Composition	♦——	This type of association represents a special case of the aggregation association. A composition indicates that a dependent (child) object instance has a mandatory association with a strong (parent) object instance. When the parent object instance is deleted, all child object instances are automatically deleted. The composition association is represented with a filled diamond in the side of the parent object instance. This is the equivalent of a weak entity in the ER model.

Examine the relationships depicted in Figure H.6 to help you understand the use of aggregation and composition.

The UML standard guides the use of the aggregation and composition constructs as follows:

- An *aggregation construct* is used when an object is composed of (or is formed by) a collection of other objects, but the objects are independent of each other. That is, the relationship can be classified as a "has a" relationship type. For example, an owner owns many cars, a team has many players, or a band has many musicians.
- A *composition construct* is used when two objects are associated in an aggregation association with a strong identifying relationship. That is, deleting the parent deletes the children instances. For example, an invoice contains invoice lines, an order contains order lines, or an employee has dependents. The use of a composition construct implies the use of the CASCADE DELETE foreign key rule in the relational database model.

FIGURE H.6 Aggregation and composition examples

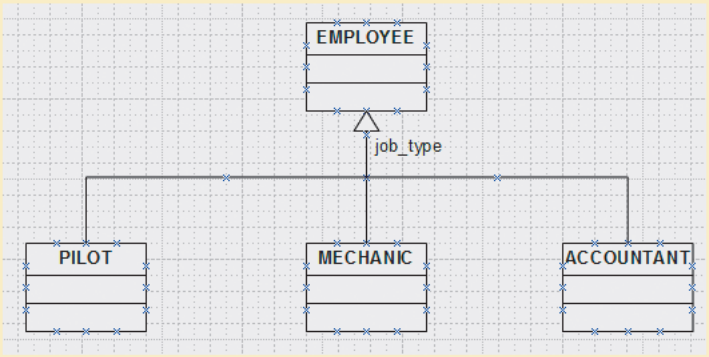


SOURCE: Course Technology/Cengage Learning

H.3 Generalizstions to Represent Supertypes and Subtype

The UML diagram also enables you to represent class generalization hierarchies in which a class "is a" subtype of another (supertype) class. You learned about those classification types in Chapter 4, Entity Relationship (ER) Modeling, and in Appendix G. Figure H.7 shows an example of a UML generalization hierarchy.

FIGURE H.7 Generalization example



SOURCE: Course Technology/Cengage Learning

The generalization hierarchy is represented by an arrow that points to the parent class. Figure H.7 shows an EMPLOYEE class supertype with three class subtypes: PILOT, MECHANIC, and ACCOUNTANT. In this case, the class hierarchy represents disjoint subtypes; that is, one employee can be related to only one subtype class (a pilot or a mechanic or an accountant).

Generalizations can also have constraints. The *job_type* label next to the generalization line indicates the EMPLOYEE attribute that was used to determine to which class subtype the instance belongs.

H.4 UML and the Relational Model

This brief tutorial has examined the use of the Unified Modeling Language (UML) Class diagram as a database design option. The main focus of UML notation is to facilitate the analysis, design, and implementation of computerized database solutions. To accomplish that task, UML uses a set of diagramming notations derived from object-oriented concepts and, in particular, object-oriented programming. Several points are worth emphasizing.

- UML is *not* a design *methodology*. It is best described as a design *notation*.
- UML notation is *not* geared specifically toward data modeling or relational database design. On the contrary, UML focuses on supporting the process of analyzing and designing information systems.
- One of UML's main characteristics is that it is extensible, thus enabling the designer to create new constructs through the use of so-called stereotypes. As used in the context of UML, a *stereotype* is a new element that represents a distinctive object, characteristic, or functionality in the model. For example, a designer can add new stereotypes to represent primary keys, foreign keys, indexes, triggers, stored procedures, views, and so on.

UML is becoming common in systems analysis and design, but *not* in database design, where relational database modeling tools such as Microsoft's Visio Professional, Computer Associates' ERwin Data Modeler, or Embarcadero's ER/Studio are the norm. Because the relational model is still the dominant data model, the adoption of a relational data modeling notation within UML would help to increase its penetration in the design and modeling market. If such a merger took place, database designers and system analysts would be able to use the same set of design tools, thus facilitating the creation of information systems.

The extensibility of UML is the characteristic that opens the door for UML to directly support relational database modeling notation. For example, as of this writing, IBM offers its Rational Rose Professional Data Modeler product, which introduces a Data Modeling Profile for UML. This tool allows database designers to create semantically rich relational database models with support for all relational constructs, such as primary keys, foreign keys, indexes, triggers, and stored procedures. Although the Data Modeling Profile is not yet a UML standard, it is backed by IBM, an active member of the OMG consortium. So this merger of concepts and tools could yield the best of both worlds.