

## Chapter 2

# Object Relationship Notation (ORN)

The *Object Relationship Notation (ORN)* was first described in Ehlmann et al. (1992). It was proposed for “representing non-inheritance relationships in an object-oriented, scientific database,” as indicated by the title of this 1992 paper. Like the relational and ER models, ORN has evolved from what was originally proposed. Its syntax, graphical representation, and semantics have changed slightly, and it has been integrated into UML class diagrams, adapted to relational databases, and applied to the development of all types of databases, not just scientific (Ehlmann and Riccardi 1994, Ehlmann and Stewart 1997, Neal and Ehlmann 2000, Ehlmann and Yu 2002, Ehlmann 2006).

Today, ORN can best be described as a declarative scheme that adds referential actions to UML multiplicities to allow the semantics of associations to be better defined. Its symbols for describing referential actions, called *bindings*, are included in ER and class diagrams to better describe associations during conceptual modeling. Its complete syntax, including multiplicities, is included in DDLs to better define associations to DBMSs, object or relational. This allows a DBMS to enforce multiplicities and perform the specified referential actions. When ORN is supported by an *ORN-extended DBMS*, a variety of association types can be easily modeled, more directly mapped to a database definition, and automatically implemented.

This chapter defines the syntax of ORN, its graphical representation in an ER and class diagram, and its semantics. It explains these semantics with a number of examples and concludes by returning to the company database model given at the end of the previous chapter to explain the ORN previewed by this example. The chapter can be read with Chapter 3, which presents a tool that is helpful in learning ORN.

## 2.1 Syntax

The syntax of ORN is that of an **<association>** and is given by the syntax diagram in Fig. 2.1. Valid syntax results only from traversing the diagram in the direction of the arrows. Some examples of **<association>**s are given below. As we shall see in Section 2.3, the last two examples are semantically equivalent.

**<1-to-\*>**    **'<5..\*-to-\*>X-**    **<0..1-to-2..15>~?**    **<0..1-to-2..15>|?X?**

Associations are described in ORN at two levels of detail. A **<multiplicities>** specification describes a binary association type solely by multiplicities, e.g., **1-to-**

1..\*, and reflects what is given in a UML class diagram. A **<multiplicity>** is given for each class or role. An **<association>** delimits a **<multiplicities>** specification by < and > symbols and adds more semantic detail by including a **<binding>** for each end of the association, e.g., !<1-to-1..\*>?.

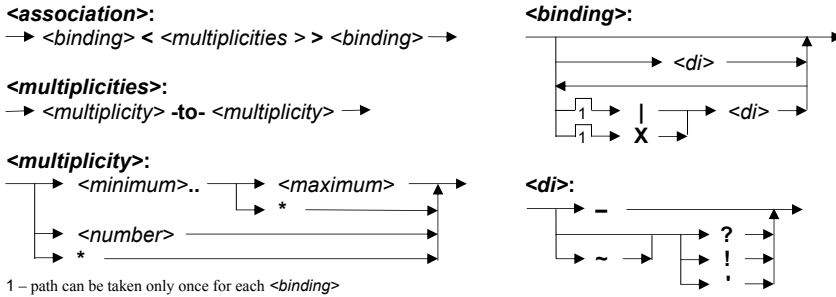


Fig. 2.1 ORN Syntax (Reprinted, with permission, from [Ehlmann 2007] © 2007 IEEE)

A **<binding>** uses just a few special characters, or symbols, to indicate the destructibility of association links, which prescribe one or more referential actions. A **<binding>** is nil, i.e., contains no symbols, which specifies *default binding*; or it is a **<di>**, *destructibility indicator*; or it is a |, an *implicit indicator*, or X, an *explicit indicator*, followed by a destructibility indicator; or it is the latter followed by the opposite of the given implicit or explicit indicator followed by another destructibility indicator. A destructibility indicator is a -, ?, !, or ' symbol, where the latter three symbols can be preceded by an optional ~, a *cascade indicator*.

## 2.2 Graphical Representation

The graphical representation of ORN makes a modest extension to the ER diagram and class diagram. Essentially a **<binding>**, which may be nil, is included at each end of a binary association. This can be done in a number of ways.

Fig. 2.2 (a) shows how **<binding>**s can be included as stereotype icons in an *ORN-extended class diagram*. In UML, a stereotype icon can be used to extend the standard notation (OMG 2005). The graphical representation of an **<association>** as given in Fig. 2.2 (a) is equivalent to the syntactic representation

**<binding1> < <multiplicity1> -to- <multiplicity2> > <binding2>**

where **<binding1>** and **<multiplicity1>** are given for the *subject class* and **<binding2>** and **<multiplicity2>** are given for the *related class*. An example of the graphical representation for a <0..1-to-2..15>|?X? association is given in Fig. 2.2 (b).

Fig. 2.3 (a) shows the |?X? binding properly placed on the association line in an *ORN-extended ER diagram*, and Fig. 2.3 (b) shows its equivalent placed next to the

entity set mapping. The former placement is appropriate when explicit and implicit indicators are given. It is also the original graphical representation for ORN in ER diagrams and is the basis for the ORN logo (see Chapter 3, bottom-left of Fig. 3.1).

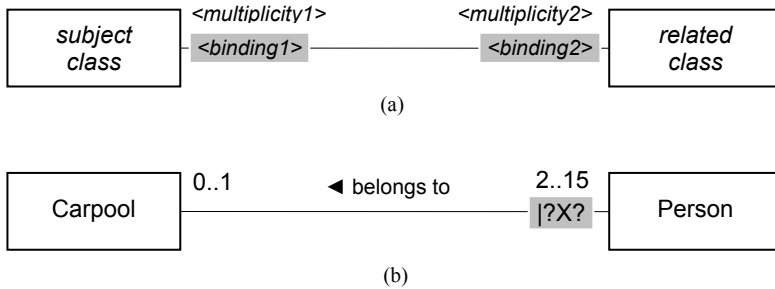


Fig. 2.2 ORN in a class diagram, (a) generic description and (b) example

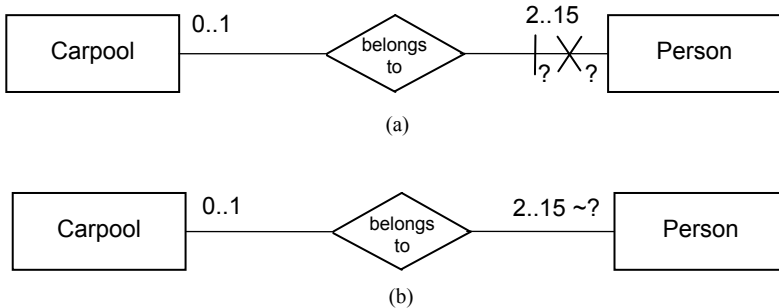


Fig. 2.3 ORN in an ER diagram, symbols placed (a) on the association line and (b) by the mappings

The class diagram representation for an **<association>** as described in Fig. 2.2 is used exclusively in the remainder of this book.

## 2.3 Semantics

ORN describes the semantics—or one could say the “nature” or “behavior”—of a large variety of binary association types. The semantics of ORN itself are derived from the semantics of the multiplicities and bindings given in an **<association>**. The role of multiplicities in describing association semantics is well known and is described in Chapter 1. The role of “bindings” needs a lot of explaining.

Bindings can be viewed three ways at three different levels.

- At the highest level, bindings when added to multiplicities determine the extent and cohesiveness of complex objects. As we saw in Chapter 1, both the relational and object models disassemble complex objects to some degree. ORN is the

“glue” that identifies and holds these objects together. The glue can provide a weak bond, a super bond as for composite objects, or somewhere in between.

- Bindings indicate the degree of “binding” between related objects by indicating the destructibility of association links, both implicit and explicit. Here, *implicit* means automatic, or system initiated, and *explicit* means application initiated, either by a program or query. *Implicit destructibility* describes whether association links involving an object can be implicitly destroyed when an object is deleted and whether such destruction requires related objects to be deleted. *Explicit destructibility* describes similar rules that apply when association links are explicitly destroyed.
- At the lowest level, bindings indicate the referential actions, or *multiplicity actions*, that should occur on object deletion and association link destruction. These system actions enforce multiplicity integrity (not just referential integrity), provide the desired implicit and explicit destructibility for an association, and implement the required implicit cascading of link destruction and deletion operations within a complex object operation.

In an  $\langle \text{association} \rangle$ , the  $\langle \text{multiplicity} \rangle$  and  $\langle \text{binding} \rangle$  before the  $\text{-to-}$  apply to the subject class, or role for an intra-class association, and those after the  $\text{-to-}$  apply to the related class or role. The subject class (or role) can be viewed as the related class (or role) and vice versa, in which case the inverse  $\langle \text{association} \rangle$  applies. In general, the inverse of  $b_1 \langle m_1 \text{-to-} m_2 \rangle b_2$  is  $b_2 \langle m_2 \text{-to-} m_1 \rangle b_1$ , and specifically that of  $\langle 0..1 \text{-to-} 2..15 \rangle \sim ?$  is  $\sim ? \langle 2..15 \text{-to-} 0..1 \rangle$ . The  $\langle \text{multiplicity} \rangle$  and  $\langle \text{binding} \rangle$  semantics for one end of an association are independent of those for the other end. Those for the other end, however, can be “in play” on certain operations, e.g., link destruction, and so can affect the final outcome of an operation.

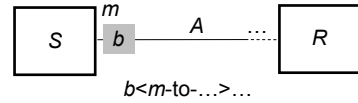
Table 2.1 gives the semantics of ORN from the perspective of a subject class  $S$  in an association  $A$ . We shall refer to this table often as we study examples of ORN that illustrate the different bindings. One thing to keep in mind with ORN semantics is that object deletion and a link destruction, implicit or explicit, can actually be implemented as object archival then deletion and link archival then destruction. In many databases today, it is rare to actually delete or destroy anything.

## 2.4 Examples

To illustrate how ORN describes association semantics, this section presents variations on the classic, normally one-to-many association between departments and employees. Instead of using “department,” however, I use “unit” since it is more generic (and shorter in length). So, the association we now focus on is: an employee “works for” a unit. Although the semantics of some of the variations on this association that we examine will seem odd, they make sense for other associations, which we explore later in this chapter and Chapter 4. Here, I focus on just this one association so the reader can focus solely on the varied multiplicities and bindings and not have to think about new associations and their own semantic peculiarities.

**Table 2.1** ORN Semantics

Semantics are given in terms of a subject class  $S$  with multiplicity  $m$  and binding  $b$  in an association  $A$  with some related class  $R$  (which could be  $S$  in a different role).



**<multiplicity>**: Semantics are the same as those in UML.

Essentially,  $m$  indicates a lower bound and upper bound on the number of objects of type  $S$  that can be related via  $A$  to each object of type  $R$ .

An  $R$  object can be created provided this does not violate  $m$ . The check for a lower bound violation is deferred until transaction commit. An  $A$  link can be created provided this does not violate  $m$ . The check for an upper bound violation is immediate. The enforcement of  $m$  on the deletion of an  $S$  object or destruction of an  $A$  link is determined by the binding  $b$ .

**<binding>**:  $A \mid b$  denotes a “cut” and an Implicit, i.e., system initiated, destruction of an existing  $A$  link that must occur on deletion of an  $S$  object. An  $X \mid b$  denotes a “cross out” and an eXplicit, i.e., user initiated, destruction of an  $A$  link.<sup>1</sup>

An  $S$  object deletion and an explicit  $A$  link destruction are *complex object operations*. Deletion of an  $S$  object succeeds only if all existing association links involving that object are implicitly destructible. Also, deletion of an  $S$  object or explicit destruction of an  $A$  link succeeds only if all required implicit object deletions succeed.

**<di>**: A destructibility indicator in  $b$  specifies the destructibility of an  $A$  link. The meaning of each indicator is given below. This meaning can alternatively be described by the actions taken on an attempt to destroy an  $A$  link. These actions are given in brackets. If a **<di>** is given after a  $\mid$ , it applies to implicit link destruction; if given after an  $X$ , it applies to explicit link destruction; and if given alone, it applies to both. If a **<di>** is not given, i.e., is nil, for implicit link destruction, explicit link destruction, or both, default destructibility applies to whichever.

- nil      *Default destructibility*. A link can be destroyed provided this does not violate  $m$ .<sup>2</sup> [Destroy the link. If  $m$  is violated<sup>2</sup>, raise an exception<sup>3</sup>.]
- *Negative destructibility*. A link cannot be destroyed. [Raise an exception<sup>3</sup>.]
- ~? or ? *Conditional cascade destructibility*. A link can be destroyed, but if this violates  $m$  (?), the destruction must be cascaded (~) to the related  $R$  object, i.e., this object must be implicitly deleted. [Destroy the link. Delete the related  $R$  object? If  $m$  is violated, yes; else no.]
- ~! or ! *Emphatic cascade destructibility*. A link can be destroyed, but the destruction must be cascaded (~) to the related  $R$  object. [Destroy the link. Delete the related  $R$  object!]
- ~' or ' *Tentative (or qualified) cascade destructibility*. A link can be destroyed, but an attempt must be made to cascade (~) the destruction to the related  $R$  object; however, this implicit  $R$  object deletion must be undone if it fails, but is required if and only if its undoing would violate  $m$ .<sup>2</sup> (Think of the ' as a “pruned back !” or as a “qualifying footnote reference” on the cascade.) [Destroy the link. Delete the related  $R$  object.' (\* – If an exception occurs on this nested complex object operation, undo the delete and then, if  $m$  is violated<sup>2</sup>, raise an exception<sup>3</sup>.)]

1 - A link change done as a single operation that replaces the  $S$  object with another is not treated as an explicit link destruction relative to class  $S$  (but is relative to  $R$ ) and is allowed if allowed by other multiplicities and bindings.

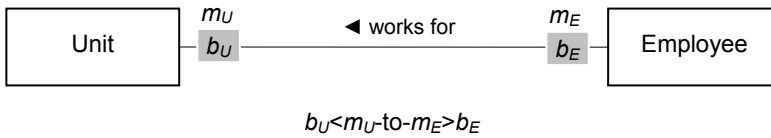
2 - The check for a lower bound violation is deferred until the end of the current complex object operation.

3 - The current complex object operation is undone.

The generic class diagram for the “works for” association is shown in Fig. 2.4 with variables given for the multiplicities and bindings. Also, included in the figure is the syntactical representation of the **<association>**, given in terms of these variables. The examples discussed in this section are created by assigning different multiplicities and bindings to these variables.

In discussing the variations on the “works for” association, we derive the precise meaning of each binding by paraphrasing from Table 2.1 while making appropriate

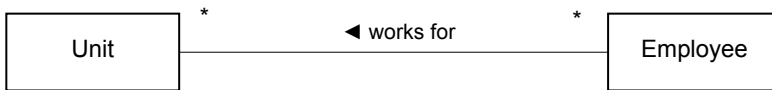
variable substitutions.  $A$  is always “works for.”  $S = \text{Unit}$ ,  $R = \text{Employee}$ ,  $m = m_U$ , and  $b = b_U$  to derive the meaning of a Unit binding. To derive the meaning of an Employee binding,  $S = \text{Employee}$ ,  $R = \text{Unit}$ ,  $m = m_E$ , and  $b = b_E$ .



**Fig. 2.4** Class diagram for “works for” association with variable multiplicities and bindings

### 2.4.1 $\langle * \text{-to-} * \rangle$

This  $\langle \text{association} \rangle$  imposes no constraints on the “works for” association. A unit can contain none or infinitely many employees, and each employee can belong to none or infinitely many units. The bindings for both classes are default since no binding symbols are given. The class diagram is shown in Fig. 2.5 (which is the diagram in Fig. 2.4 where  $m_U = m_E = *$  and  $b_U = b_E = \text{nil}$ ).



**Fig. 2.5** Class diagram for a  $\langle * \text{-to-} * \rangle$  “works for” association

The *default binding* for Unit implies both *implicit default binding* and *explicit default binding*. ORN semantics require that an object cannot be deleted unless all associations that it has with other objects can be implicitly destroyed. This means that when an object of type Unit is deleted, all links that it has with other objects, including any Employee objects, must be implicitly destructible. The implicit binding for a deleted object’s class, e.g., Unit, in an association, e.g., “works for,” determines whether such implicit destruction of the links of the association type is allowed.

Thus, the implicit default binding for the Unit class in the “works for” association applies when a Unit object is deleted and means (paraphrasing from the nil case for a destructibility indicator in Table 2.1): a “works for” link can be destroyed provided this does not violate the  $*$  multiplicity. A  $*$ , i.e.,  $0..*$ , multiplicity is never violated. Thus, a unit can always be deleted, even if it has some employees (unless the deletion is disallowed by other associations in which it is involved). The referential actions given for the default binding in Table 2.1 make it clear (if it was not already) that deletion of a unit results only in the implicit destruction of the “works for” links with related employees; the related employees are retained.

The explicit default binding for **Unit** applies when a “works for” link is explicitly destroyed and means (again, paraphrasing from Table 2.1): the link can be destroyed provided this does not violate the \* multiplicity. Again, a \* is never violated. Thus, a “works for” link can always be explicitly destroyed, at least based on the **Unit** multiplicity and binding. For explicit destruction, however, the multiplicities and bindings at both ends of an association are applicable.

The implicit default binding for the **Employee** class in the “works for” association applies when an **Employee** object is deleted and means (paraphrasing from Table 2.1, now where  $S = \text{Employee}$  and  $R = \text{Unit}$ ): a “works for” link can be destroyed provided this does not violate the \* multiplicity. A \* is never violated. Thus, an employee can always be deleted.

The explicit default binding for **Employee** applies when a “works for” link is explicitly destroyed and means (paraphrasing from Table 2.1): the link can be destroyed provided this does not violate the \* multiplicity. A \* is never violated. Thus, a “works for” link can always be explicitly destroyed, based on the multiplicities and explicit bindings at both ends of the association.

## 2.4.2 <1-to-\*>

This *<association>* may be the most appropriate description for the “works for” association. It is the one shown in the class diagram of Fig. 1.22. A unit can contain any number of employees but each employee must belong to one and only one unit. The association exemplifies the use of a default binding with a constrained multiplicity lower bound and upper bound. The class diagram is shown in Fig. 2.6 (which is the diagram in Fig. 2.4 where  $m_U = 1$ ,  $m_E = *$ , and  $b_U = b_E = \text{nil}$ ).



**Fig. 2.6** Class diagram for a <1-to-\*> “works for” association

The 1, or more precisely the 1..1, multiplicity for **Unit** in this association imposes constraints on object and link creation. Paraphrasing the second paragraph under *<multiplicity>* in Table 2.1: an **Employee** object can be created provided this does not violate the 1 multiplicity. The check for a lower bound violation (of 1) is *deferred* until transaction commit (i.e., delayed until the commit of the application-defined transaction). A “works for” link can be created provided this does not violate the 1 multiplicity. The check for an upper bound violation (of 1) is *immediate* (i.e., done as part of the current operation). Thus, an employee cannot be created unless it is linked to a unit, and an employee cannot be linked to more than one unit.

The implicit default binding for the **Unit** class in the “works for” association applies when a **Unit** object is deleted and means (paraphrasing from Table 2.1): a

“works for” link can be destroyed provided this does not violate the 1 multiplicity. The 1 is violated if the **Unit** object being deleted is linked to an **Employee** object because an employee must belong to a unit. Thus, a unit cannot be deleted if it has any employees. It can, however, be deleted if it has no employees (at least based on this association).

The explicit default binding for **Unit** applies when a “works for” link is explicitly destroyed and means (paraphrasing from Table 2.1): the link can be destroyed provided this does not violate the 1 multiplicity. Here again, the 1 is violated because an employee must belong to a unit. Thus an association between a unit and an employee, once created, cannot be explicitly destroyed.

It can, however, be changed. Paraphrasing the fine print of footnote 1 in Table 2.1: A link change done as a single operation that replaces the **Unit** object with another is not treated as an explicit link destruction relative to the **Unit** class (but is relative to the **Employee** class) and is allowed if allowed by other multiplicities and bindings. We shall soon see that this link change is allowed by the multiplicity and binding of the **Employee** class, for which the link change is treated as an explicit link destruction (since a unit is losing an employee and another is gaining one). Thus an employee who works for a unit can be assigned to another unit.

The implicit and explicit default bindings for the **Employee** class mean the same as they did in the <\*-to-\*> association. Explicit destruction of a “works for” link is allowed, at least based on \* multiplicity and default binding for **Employee**. While this allows the link change discussed in the previous paragraph, explicit destruction of a “works for” link is stymied by the 1 multiplicity and default binding at the **Unit** end of the association. By the way, a link change that exchanges one employee for another is disallowed (see again footnote 1). Allowing such would mean an employee would be left without a unit.

Since both footnotes 2 and 3 of Table 2.1 are referenced for default destructibility and are relevant to a 1 multiplicity, I now explain the last of the fine print. I do so in the context of the <1-to-\*> “works for” association and an attempted deletion of the **Unit** object.

With ORN an explicit object deletion is a *complex object operation*, as is any explicit link destruction or change. Such an operation takes place within its own internal transaction and can involve many implicit link destructions and object deletions based on the ORN semantics of one or more associations. For a **Unit** object deletion, footnote 2 means that the check for a lower bound violation of the 1, i.e., 1..1, multiplicity is not done until the end of the transaction that encompasses this complex object operation, i.e., it is partially deferred. In most cases, deferred versus immediate checking is immaterial in understanding ORN semantics, but in some unusual cases, it can make a difference. For example, in deleting a **Unit** object it is possible—though very unlikely with this association—that because of other associations and bindings, an **Employee** object that was linked to the **Unit** object at the beginning of the complex object deletion has been implicitly deleted by the time of its conclusion. If so, deferred checking at the commit of the operation will not reveal a violation of the 1 multiplicity for this deleted **Employee** object. Thus, its original link to the deleted **Unit** object will not cause the failure of the explicit **Unit** object deletion.



Footnote 3 states that the complex object operation is undone, which means the internal transaction that encompasses the operation is rolled back. Thus, when the deletion of a **Unit** fails because, for instance, the 1 multiplicity for **Unit** in the “works for” association is violated at the end of this complex operation, all implicit link destructions and object deletions resulting from the operation, as well as the original explicit **Unit** deletion itself, are rolled back.

As we have seen, the semantics of default bindings simply enforce multiplicities. Next, we examine the non-default bindings.

### 2.4.3 <0..1-to-\*>|-

With this <association> a unit can have none or many employees and each employee may (or may not) work for a single unit. An implicit negative destructibility binding is given for **Employee**. The class diagram is shown in Fig. 2.7 (which is the diagram in Fig. 2.4 where  $m_U = 0..1$ ,  $m_E = *$ , and  $b_U = \text{nil}$ ,  $b_E = |-$ ).

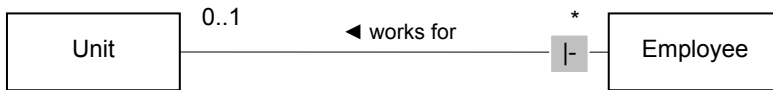


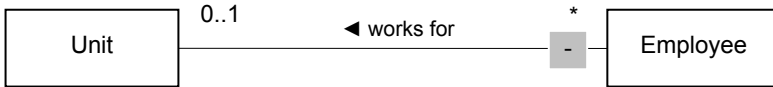
Fig. 2.7 Class diagram for a <0..1-to-\*>|- “works for” association

The  $|-$  binding for **Employee** in the “works for” association applies when an **Employee** object is deleted and means (paraphrasing from Table 2.1): a “works for” link cannot be destroyed. Thus, since ORN dictates that an object cannot be deleted unless all existing links involving the object can be implicitly destroyed, an employee who works for a unit cannot be deleted. Such an employee’s link with a unit would have to be explicitly destroyed, which is now allowed by the 0..1 multiplicity and explicit default binding, before the employee could be deleted.

### 2.4.4 <0..1-to-\*>|-X-

This <association> is like the previous one except that an explicit negative destructibility binding is given for the **Employee** class. Since both implicit and explicit negative destructibility are specified, the association can also be described as <0..1-to-\*>-|. Its class diagram is shown in Fig. 2.8 (which is the diagram in Fig. 2.4 where  $m_U = 0..1$ ,  $m_E = *$ , and  $b_U = \text{nil}$ ,  $b_E = |-X-$  or  $-|$ ).

The  $X-$  binding for **Employee** in the “works for” association applies when a “works for” link is explicitly destroyed and means (from Table 2.1): a “works for” link cannot be destroyed. Thus, an association between a unit and an employee, once created, cannot be explicitly destroyed.



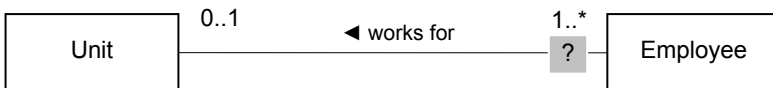
**Fig. 2.8** Class diagram for a  $\langle 0..1\text{-to-}*\rangle$ - “works for” association

It can, however, be changed. For the X- binding, the variable *S* is *Employee* and *R* is *Unit*, and we again paraphrase footnote 1 in Table 2.1: a link change done as a single operation that replaces the *Employee* object with another is not treated as an explicit link destruction relative to the *Employee* class (but is relative to the *Unit* class) and is allowed if allowed by other multiplicities and bindings. This link change is allowed by the multiplicity and binding of the *Unit* class, for which the link change is treated as an explicit link destruction. Thus an employee who works for a unit can be replaced by another employee.

This link change would not be allowed if we add the X- binding to the *Unit* end of this association. An X- $\langle 0..1\text{-to-}*\rangle$ - association joins an employee with a unit until “death do ye part.” Here, this means until the unit is deleted because the |- binding for *Employee* keeps a linked employee from being deleted. (The link change not allowed by the X- $\langle 0..1\text{-to-}*\rangle$ - association would also not be allowed by a  $\langle 1\text{-to-}*\rangle$ - association, because an employee would be left without a unit.)

### 2.4.5 $\langle 0..1\text{-to-}1..*\rangle$ ?

This *<association>* specifies that each unit must have at least one worker. It exemplifies the use of conditional cascade destructibility with a lower bound multiplicity constraint. Since both |? and X? are implied by the ?, the association can also be described as  $\langle 0..1\text{-to-}1..*\rangle|?X?$ , or  $\langle 0..1\text{-to-}1..*\rangle|~?X~?$  if the optional cascade symbol is given. The class diagram for this association is shown in Fig. 2.9.

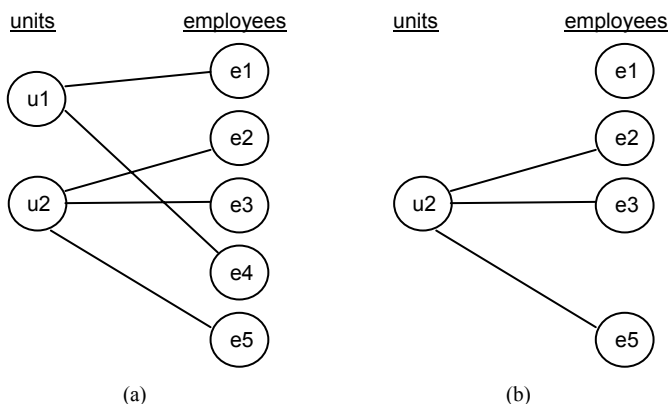


**Fig. 2.9** Class diagram for a  $\langle 0..1\text{-to-}1..*\rangle$ ? “works for” association

The |? binding for the *Employee* class in the “works for” association applies when an *Employee* object is deleted and means (paraphrasing from Table 2.1): a “works for” link can be destroyed, but if this violates the multiplicity  $1..*$ , the destruction must be cascaded to the related *Unit* object, i.e., this object must be implicitly deleted. The 1 is violated if the *Employee* object being deleted is the only one linked to the related *Unit* object. Thus, deletion of the last employee who works for a unit causes the deletion of the unit.

The X? binding applies when a “works for” link is explicitly destroyed and means (from Table 2.1): a “works for” link can be destroyed, but if this violates the multiplicity 1..\*, the destruction must be cascaded to the related Unit object, i.e., this object must be implicitly deleted. Again, the 1 is violated when the Employee object being “de-linked” from the Unit object is the only one linked to this object. Thus terminating the “works for” relationship between a unit and its last employee terminates the unit!

If the association was  $\langle 0..1\text{-to-}2..*\rangle?$ , then deletion or explicit delinking of one of just two employees working for a unit would cause deletion of the unit. Assuming this association, Fig. 2.10 shows object diagrams representing the state of a database before and after the deletion of an employee e4. In an object diagram, circles represent objects and lines represent the association links between these objects.



**Fig. 2.10** For a  $\langle 0..1\text{-to-}2..*\rangle?$  association between units and employees, object diagrams for the database state (a) before and (b) after deletion of e4

### 2.4.6 $! \langle 0..1\text{-to-}*\rangle$

This  $\langle \text{association} \rangle$  exemplifies the use of emphatic cascade destructibility. Since both  $!$  and  $X!$  are implied by the  $!$ , the association can also be described as  $!X! \langle 0..1\text{-to-}*\rangle$ , or  $|\sim!X\sim! \langle 0..1\text{-to-}*\rangle$  if the optional cascade symbol is given. The class diagram for this association is shown in Fig. 2.11.



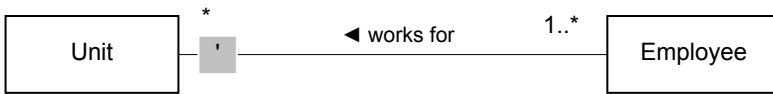
**Fig. 2.11** Class diagram for a  $! \langle 0..1\text{-to-}*\rangle$  “works for” association

The **|!** binding for the **Unit** class in the “works for” association applies when a **Unit** object is deleted and means (from Table 2.1): a “works for” link can be destroyed, but the destruction must be cascaded to the related **Employee** object (i.e., this related object must be implicitly deleted). Thus, deletion of a unit causes the implicit deletion of all employees who work for that unit.

The **X!** binding applies when a “works for” link is explicitly destroyed and means (from Table 2.1): the “works for” link can be destroyed, but the destruction must be cascaded to the related **Employee** object (i.e., this related object must be implicitly deleted). Thus terminating the “works for” relationship between a unit and an employee always terminates the employee.

#### 2.4.7 '<\*-to-1..\*>

This **<association>** exemplifies the use of tentative cascade destructibility. It allows an employee to work for many units but each unit must have at least one employee. Since both **|** and **X** are implied by the **'**, the association can also be described as **|X'<\*-to-1..\*>**. The class diagram for this association is shown in Fig. 2.12.



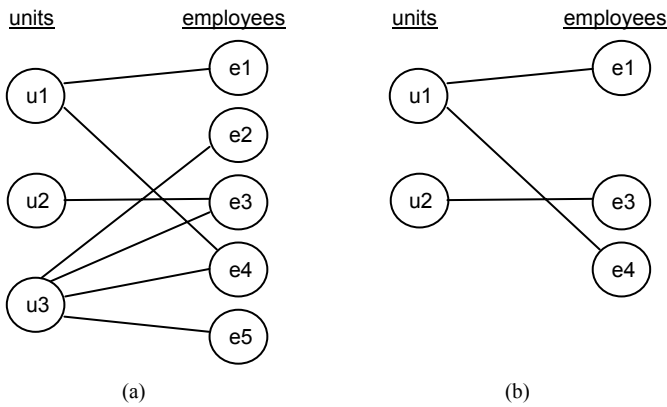
**Fig. 2.12** Class diagram for a '**<\*-to-1..\*>** “works for” association

The **|** binding for the **Unit** class in the “works for” association applies when a **Unit** object is deleted and means (from Table 2.1): a “works for” link can be destroyed, but an attempt must be made to cascade the destruction to the related **Employee** object; however, this implicit **Employee** object deletion must be undone if it fails, but is required if and only if its undoing would violate the **\*** multiplicity. A **\*** is never violated, so the deletion of the related **Employee** object is never required. Here, the attempted deletion of a related object fails if it is the only **Employee** object related to another **Unit** object. It fails because of the lower bound **1** multiplicity and default implicit binding for the **Employee** class, which does not permit the implicit destruction of an **existing** link when the multiplicity is violated. Thus, deletion of a unit causes the implicit deletion of all employees who work for that unit except any such employee who is the only employee working for another unit.

The **X** binding applies to the explicit destruction of a “works for” link. Its meaning is similar to that of the **|** binding. Essentially, on explicit destruction of a link, the related **Employee** object is implicitly deleted unless it is the only **Employee** object linked to another **Unit** object.

Let us suppose that on deletion of a unit or explicit destruction of an employee’s working relationship with a unit, we do not want to delete **any** employee who is

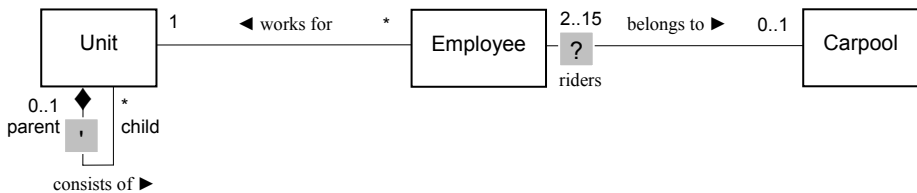
working for another unit. Then, the `<association>` should be `'<*-to-1..*>|-`. Now, the `|-` binding for the **Employee** class blocks the implicit destruction of an existing link on an attempted deletion of an employee, thus making it fail. Employees, however, who do not have existing links with other units are deleted. Assuming this association, Fig. 2.13 shows object diagrams representing the state of a database before and after the deletion of a unit `u3`. The “blockage” provided by the `|-` is independent of the **Employee** multiplicity and so would also work for a `'<*-to-*>|-` association between units and employees.



**Fig. 2.13** For a `'<*-to-1..*>|-` association between units and employees, object diagrams for the database state (a) before and (b) after deletion of `u3`

## 2.5 Flashback to the Company Database

We now revisit the company database example given at the end of Chapter 1. Fig. 2.14 shows the same extended class diagram that was given in Fig. 1.24 (a). The extensions to the diagram—i.e., the ORN bindings, both explicit and default—can now be examined and understood based on ORN semantics presented in this chapter.



**Fig. 2.14** ORN-extended class diagram for part of a company database

Repeated below are the descriptions of relevant association semantics for this application that could not be modeled in the standard class diagram. The applicable bindings and multiplicities that represent each semantic in the extended-ORN class diagram are given within parentheses. Also given within these parentheses is a reference to a subsection of Section 2.4 that describes a semantically similar variation of the “works for” association between employees and units. Hopefully now, after a bit of study, how these nontrivial semantics can be represented in a class diagram by simply adding two symbols—' and ?—will be clear to the reader.

- If an attempt is made to explicitly remove a unit that has employees working for it, an error should result because all employees must work for a unit (the implicit default binding and 1 multiplicity for **Unit** in the “works for” association, see Section 2.4.2).
- If a unit is removed, all subordinate, i.e., component or **child**, units should be automatically removed unless those units still have employees working for them (the |' binding for the **parent** end of the “consists of” association along with the implicit default binding and 1 multiplicity for **Unit** in the “works for” association, see Section 2.4.7). Such units may remain independent (the 0..1 multiplicity for the **parent** end of the “consists of” association) or later be placed within another unit.
- If a unit is removed from its parent unit, similar semantics should apply. It and all of its subordinate units should be automatically removed unless they have employees working for them (the X' binding for the **parent** end of the “consists of” association along with the implicit default binding and 1 multiplicity for **Unit** in the “works for” association, see Section 2.4.7).
- A carpool is defined by having at least two riders, so that if the number of riders falls below two, either because of employee terminations or because employees quit the carpool, the carpool really no longer exists and so should be automatically removed (the ? binding and 2..15 multiplicity for **Employee** in the “belongs to” association, see Section 2.4.5).

The next chapter discusses a tool that allows the reader to very easily model all of the associations presented in this chapter using ORN and to readily observe their semantics being automatically enforced by a database system. The visualization provided by this type of tool is intended to help the user become competent at using ORN.

Object Relationship Notation (ORN) for Database  
Applications

Enhancing the Modeling and Implementation of  
Associations

Ehlmann, B.K.

2009, XXII, 246 p. 173 illus., Hardcover

ISBN: 978-0-387-09553-0